ON THE DEVELOPMENT AND MANAGEMENT OF ADAPTIVE BUSINESS COLLABORATIONS

OVER HET ONTWIKKELEN EN ONDERHOUDEN VAN ADAPTIEVE BEDRIJFSSAMENWERKINGEN

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit van Tilburg, op gezag van de rector magnificus, prof. dr. F.A. van der Duyn Schouten, in het openbaar te verdedigen ten overstaan van een door het college voor promoties aangewezen commissie in de aula van de Universiteit op woensdag 12 september 2007 om 14.15 uur

door

Bart Orriëns

geboren op 4 november 1978 te Doetinchem.

Promotor: prof. dr. M. P. Papazoglou
Copromotor: dr. J. Yang

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems, and CentER, the Graduate School of the Faculty of Economics and Business Administration of Tilburg University.

To my family

Contents

List of Figures

Preface

He who chooses the beginning of a road chooses the place it leads to. It is the means that determine the end; Harry Emerson Fosdick

Great is the art of beginning, but greater is the art of ending; Henry Wadsworth Longfellow

Sometimes it seems like an eternity, but I still remember the day when I first got acquainted with the mysterious and mesmerizing world of web services and service-oriented computing. The year was 2001 and I was looking for a master thesis project for my studies Information Management at Tilburg University. I had been searching for a while when I, thanks to two friends of mine, came in contact with Jian Yang from (then) Tilburg University. She mentioned the, at that time, cutting edge new technology of web services, and convinced me to do my thesis on the intricacies of e-business registries to publish and find such web services. My task was to examine current proposals for such registries, compare them and extrapolate their generic characteristics and requirements. The result was a lengthy piece of ninety-three pages, something only to be outdone many years later by the work that lies before you now.

After this short but intense six months of exposure to this area of computer science, I moved on to study subjects in the area of Computational Linguistics and Intelligence. This proved to be a short detour however, as by the beginning of 2002 I found myself involved in the PIEC project, a collaboration between the Department Of Information Management at Tilburg University and the Telematica Institute on the integration of legacy applications through the usage of web services. My job was to implement and evaluate a proposal made by Jian Yang and Mike Papazoglou to glue web services together into more complex ones through a process called web service composition, as such achieving higher-level functionality that was usable in the context of business processes. While at the same time I was still pursuing Artificial Intelligence and Language Recognition, I found myself more and more immersed in the service-oriented computing domain and the potential it had to offer.

As such, in the end it came as no surprise that when the PIEC project had finished and Jian asked me to pursue a PhD in this direction, I almost immediately accepted her offer. Having been exposed to the realm of scientific research for some time already, I found it very appealing to try and attempt to contribute something to the then novel and largely unexplored research domain of service composition. Together with Jian as well as Mike

(who by then had become my official promotor) we decided on investigating the possibilities of dynamically composing services in accordance with a given set of requirements. I will forego here on the details on all great things that such dynamic composition has to offer (the following three hundred pages should attest to that), but let me just say that this was the start of a four year long exploration in which the scope of the research seemed to increase by the week. I am indebted therefore to both Jian and Mike for making sure that in the end I didn't try to do everything that there is to do in the world of dynamic and service-oriented computing based business collaboration.

And now it is done. Four years of research have culminated into a reasonably thick book with an extensive and sometimes to me mindboggling approach for rule based dynamic business collaboration development and management. Along the way I have studied countless works on the most diverse range of topics, established my own approach based on the strengths and weaknesses of these works, and contributed a number of papers myself to the ever growing paper pile. This dissertation, if I do say so myself, makes a nice final contribution. My PhD also gave me the chance to visit places I would most likely not have seen otherwise. Conferences, it seems, are always held at the most lovely locations like Chicago, Rome and Xi'an. A place that has really captured my heart is Sydney, Australia. I am forever grateful to Jian for inviting me (twice) to come over and work with her. Not only did those fourteen months do wonders for the progress of my research, but they also gave me the chance to explore Sydney and Australia, and New Zealand, and to come in contact with so many nice people which I feel very lucky to call my friends today.

The accomplishment of this work would not have been possible without the guidance and support of many people. The fact that this book exists at all must first of all be contributed to my promotor Mike Papazoglou. Our meetings were always illuminating and made sure that I did not stray too far from the task I had set out to accomplish. As to the extent in which this dissertation presents valuable ideas with scientific merit, such credit foremost goes to my co-promotor Jian Yang. She has patiently supervised and coached me over the years, made sure I stayed focused, provided invaluable suggestions, remarks and corrections, managed to convince me that there was a meaning to everything that I was doing, and, most important of all, was able to put up with both my writing and the fact that I so often changed things seemingly overnight. Many thanks also go to my colleagues, Benedikt, Frans, Jeroen, Kees, Vasilios and Michele at INFOLAB at Tilburg University for the ideas they contributed in numerous debates and discussions, and for providing such a pleasant and stimulating environment to work in. My appreciation also goes to the members of my PhD committee consisting of Hans Weigand, Manfred Jeusfeld and Fabio Casati. They were kind enough to study my ideas, evaluate their merits as well as their weaknesses, and contribute valuable feedback.

Of course staying sane while working on a specialized area such as rule based service-oriented computing based business collaboration is not an easy endeavor, and therefore I am very grateful to my friends who have made sure that I didn't go entirely crazy: Toine, Marilene, Marie-Jose, Carina, Antonie, Martijn, Femke, Gerrit-Jan and Armand, thanks for everything. Many thanks also to everybody in the lunch group: Buttercup, Blossom, Bubbles, Beatmaster K, doctor Klootzak, Momo, V (or whatever you want to call me),

the Cappucino Man, the Banana King, the Sexy Guru and Willem II. So many wonderful lunches, coffee breaks, trips, and much more, it is an honor to be your Dutch Guru. My unabridged love and affection go to Gema for her support, especially in the last phase of the dissertation. It is extremely rare to find a 'tanghe' partner and I consider myself very blessed to have found you. Finally, my thanks goes to the support from my family whose love has been a source of comfort and inspiration throughout my life. Words cannot and will never be able to express my gratitude. Over the years you have sometimes heard me utter the most strange expressions and from time to time showed you completely unrecognizable scribbles in a vain attempt to explain what I was doing. Perhaps it will all become clear when you read this small booklet. It is just little over three hundred pages long, so that should not be too difficult to do. Especially if you know that to you each and every word is dedicated.

Bart Orriëns, September 12, 2007

Chapter 1

Introduction

There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success than to take the lead in the introduction of a new order of things; Niccolo Machiavelli

The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in, we're computer professionals. We cause accidents; Nathaniel Borenstein

Todays business climate demands a high rate of change with which Information Technology (IT)-minded organizations are required to cope. Organizations face rapidly changing market conditions, new competitive pressures, new regulatory fiats that demand compliance, and new competitive threats. All of these situations and more drive the need for the IT infrastructure of an organization to respond quickly in support of new business models and requirements. Only in this way can an organization gear towards the world of semi-automated, complex electronic transactions. *Business collaboration* is about cooperation between organizations by linking their business processes and exchanging messages in order to achieve some shared goal(s). Fig. 1.1 shows an example of such business collaboration in which several parties are working together in order to facilitate the efficient handling of insurance claims.

Based on the case study in (Grefen et al., 2000) (described in full in Appendix A), the figure shows there are multiple businesses involved in the cooperation such as `AGFIL` responsible for underwriting the motor policy and covering losses incurred, `Lee Consulting Services (Lee C.S)` responsible for coordinating and managing the operation of the emergency service on a daily basis, `Europ Assist` responsible for handling and recording policyholder phone calls, `Garages` responsible for providing courtesy cars and repair service, and `Assessors` responsible for inspecting cars and double-checking car repair estimates. If we look for a moment in this example from the point of view of `Lee C.S` we find that its activities involve among others exchanging information with `AGFIL`, negotiating and approving car repair costs with `Garages` and involving `Assessors` to inspect cars. As such, `Lee C.S` has multiple bi-lateral business collaborations with the other parties involved in the claim handling process. In a world of semi-automated transactions such interactions

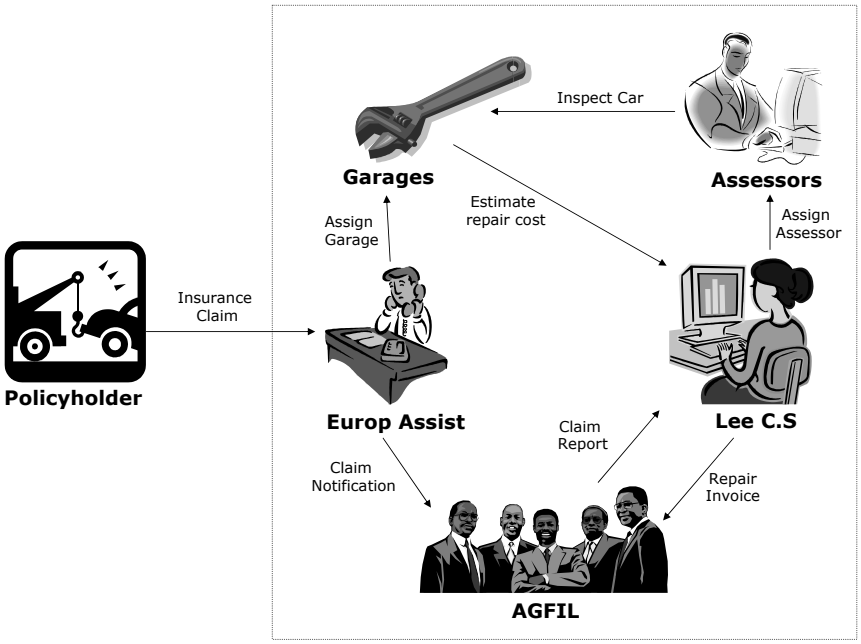

Figure 1.1: The AGFIL Business Collaboration

require the various IT-systems of the different organizations to exchange messages. The realization of such message exchange by itself is a challenge as it requires the adoption of appropriate standards and technologies. For example, `Lee C.S` will require the means to exchange claim information with `AGFIL` as well as communicate car repair estimate approvals with `Garages`.

Equally (if not more) important though is that the parties will need to align their business processes and policies to ensure that their activities are carried out conform their respective objectives. To illustrate, it is quite feasible that from a technical point of view `Lee C.S` and `AGFIL` can exchange messages to communicate insurance claim information. However, if they can not agree on when to exchange what documents, then no collaboration will take place. Alternatively it may be the case that depending on a specific type of claim (with particular conditions) `Lee C.S` and `AGFIL` will interact through the usage of different IT systems in order to exchange various kinds of information. In such case higher level business considerations directly influence how messages are exchanged on an IT-system level. To further complicate matters it is not uncommon in the current business environment for business processes and policies to rapidly change. In such events organizations must be able to assess the impact of these changes on their existing business collaborations both from a business and technical perspective. For example, if `Lee C.S` wants to change when an external `Assessor` must be contacted to inspect a car, then it will need to adjust its IT systems.

A final complication is the fact that although each bi-lateral collaboration between `Lee`

C.S and the other parties can be viewed as a separate cooperation, they are at the same time interrelated as the results of one can affect the other collaborations. To illustrate, when Lee C.S decides to adopt a new policy regarding when to call in an Assessor, it will communicate this updated policy to the different Garages. Each Garage must then be able to evaluate the effects of the new policy on their internal repair process and its interactions with the other parties. In the case of this specific change the private routine of Garage concerning when to ask for repair approval to Lee C.S and when to just proceed with the repair will require adjustment. Otherwise, it may occur that Lee C.S hires an Assessor to inspect a car while at the same time the Garage already started performing repairs. Such interdependencies make that the building and managing of business collaborations that cross independent organizational boundaries and their IT systems is challenging for parties like Lee C.S as it requires them to link their own activities with those of the other businesses together into a cohesive and meaningful whole. Put differently this means that companies such as Lee C.S must be able to conduct their activities in correspondence with their own policies and routines while at the same time adhere to all the agreements made with the different partners.

In order to enable organizations to accomplish the development and management of business collaborations like the one sketched above in a dynamic fashion while adhering to the requirements imposed by the business environment, they must be capable of properly capturing, modeling and managing the specifics of their business collaborations. To this end business collaboration design needs to apply software development principles and at the same time incorporate the special requirements of modern business collaboration development, i.e, support for high (abstract) level specification and adaptive to market changes. This requires modeling languages, methodologies, techniques and tools that allow designers to rapidly develop and deliver business collaboration designs based on proven and tested models. Furthermore, new designs must be verifiable to determine if the modeled collaborations are in accordance with current market conditions, government regulations, industry guidelines, internal policies and so on. Similarly, modifications to existing designs must be assessable to check that resulting collaborations remain compliant. This requires the modeling languages to facilitate the specification of both business and technical demands for a business collaboration, as well as dependencies among them. It also requires these languages to support the definition of private business processes, public business collaborations, and any dependencies among them.

A technology for realizing dynamic business collaborations that is becoming more and more popular is service-oriented computing (SOC) based middleware. SOC is a paradigm for distributed computing and e-business processing based on the notion of IT systems providing services in a standardized and platform independent manner. As such, SOC enables the integration of heterogeneous systems and applications possibly belonging to different organizations. Moreover, SOC promises the realization of business collaborations in the most optimal manner by selecting and combining the most suitable and economical services, where these services may be self maintained or offered by other organizations. A business collaboration development in the context of SOC normally involves: (1) the development of private *business processes* using service orchestration language such as

BPEL4WS (Curbera et al., 2002); (2) the development of the *business protocols* to interact with other business partners (the protocol is also often called the public behavior of a business or abstract business model); (3) finally the development of *business agreements* for the business collaboration via usage of service choreography. The challenge during such development is how to ensure and maintain consistency for each partner in the collaboration as well as consistency for the collaboration as a whole in the presence of changes as analyzed above. Consistency in this regard can informally be thought of as the lack of contradictions in the behavior of each individual partner as well as the lack of contradictions between the behaviors of the different partners. For example, `Lee C.S` will want to assure that the different `Garages` seek car repair approval in the same situations as those in which `Lee C.S` will hire external `Assessors`.

Unfortunately, current SOC based business collaboration development and management solutions including the defacto standard BPEL4WS (Curbera et al., 2002) are too narrowly focused. The approach that these solutions take is technical in nature, where they provide the means for defining, generating and composing web services to implement business collaborations. However, defining, generating, and managing such cooperative processes is much more difficult than simply defining and generating web services. In developing business collaboration, technology is secondary to the policy, business rules, information, and processes that use and create the information and the services. Without the proper means to capture the relation between the policies, business rules, information and processes of an organization and the manner in which the organization uses its services, it is very difficult to determine whether developed business collaborations are in accordance with the requirements from the business environment. Linking back to the example in Fig. 1.1 for a moment, it is not really feasible with current SOC based solutions for `Lee C.S` to capture a situation in which exchanging information about different types of insurance claim lead to different IT-system level message exchanges. Also, it is very difficult for `Lee C.S` to determine whether its conditions concerning when to involve `Assessors` are consistent with those followed by `Garages`.

With solutions like offered by ebXML (ebXML Initiative, 2006) and UML based notations such as presented in (Booch et al., 1998) organizations can capture business level requirements. However, these solutions typically lack the means to explicitly capture the corresponding lower level technical realization in terms of the services used (i.e. the messages exchanged). They are also limited in their support for capturing policies and business rules, which typically are often subject to change in organizations. It would be difficult therefore for `Lee C.S` to adequately capture the existence of different insurance claim information demands in Fig. 1.1, for `Lee C.S` and `AGFIL` to express how those demands lead to multiple unique and distinctive lower level message exchanges, and for `AGFIL` to assess the impact of changes to the informational demands on its cooperations with the other parties. Furthermore, service composition based on these (as well as on the aforementioned web service based) technologies and standards is very much a manual process. As such, when the requirements imposed by the agile and dynamic business environment change (for example the aforementioned change in `Lee C.S`'s policy as when to hire an external assessor), it becomes a costly and time-consuming affair in current composite web service

development and management solutions to determine the impact of these changes on new and existing SOC based business collaborations; both in terms of their compliance with the modified requirements as well as the consistency of the resulting designs.

Our goal in this dissertation is to provide an environment in which the people who develop and manage business collaborations in organizations can do so in a way that is as independent of specific SOC implementation technologies as possible, where they can take business requirements into consideration, and in which they can respond to any changes as effectively as possible. In the remainder of this chapter we provide a global overview of our research. We first give a background overview in section 1.1 to place the presented research into context. Then, section 1.2 provides motivation for the presented research, where requirements and open issues are identified. Subsequently, the proposed solution to address the identified requirements and issues is outlined in section 1.3. After that the overall research goal and objectives are stated in section 1.4. The scope of the research is then defined in section 1.5. The main research questions are next stipulated in 1.6. Section 1.7 gives an overview of the followed research methodology. The chapter concludes in section 1.8 with a presentation of the structural outline of the dissertation.

1.1 Research Background

As we indicated in the introduction our research is concerned with dynamic service-oriented business collaboration development and management. To sketch the context of the research presented in this dissertation we provide background information in this section on its three main research topics: business collaboration, change, and service-oriented computing. We first introduce the notion of business collaboration in section 1.1.1. Subsequently, we discuss the role of change in business collaboration and the requirements it poses for business collaboration development and management in section 1.1.2. After that in section 1.1.3 we focus on the emerging paradigm of service-oriented computing and argue its usefulness for dynamic business collaboration.

1.1.1 Business Collaboration

It is said that no man is an island and nor is the modern organization. Rather, organizations are typically part of complex inter-organizational structures such as the complex web illustrated in Fig. 1.1. In this dissertation business collaboration encompasses these forms of cooperation and can be informally thought of as a cooperation between organizations working together to achieve some shared business goal(s). Internally in such cooperations each organization performs its own business processes to further the accomplishment of these goals. These lead to the exchange of information with other organizations, which in turn influence the processes of these parties. Business collaborations are often referred to as *inter-organizational processes* (or public processes), whereas business processes are often called *intra-organizational processes* (or private processes). These two types of process are intricately interconnected as intra-organizational processes both support and are triggered

by inter-organizational processes. In addition, processes are not one-dimensional in nature. Rather, they span the strategies, goals and plans, the operational activities, and the information systems, applications, and etceteras of organizations. As such, organizations must be able to work together both from a business and technical point of view (also argued for in e.g. (Bresciani et al., 2004), (Jonkers et al., 2003) and (Nagy et al., 2004)). Moreover, both private and public processes are influenced by many factors within the organization such as its employees, available resources and supported activities.

To make their processes more efficient organizations have been long occupied with *business process automation*, that is, having their business processes execute with the help of IT systems. Business process automation is usually divided into two phases: development and enactment. During development (also called design time) the to-be-automated process is analyzed via *business process design* and a model of the process is created accordingly. This design provides an (often simplified) process view based upon which individual processes can be executed (i.e. enacted. Such instantiations of a design are called *business process instances*, and they are utilized during enactment (also called runtime) whenever the process is to be executed. In addition, organizations have adopted IT-based solutions to coordinate the interactions between their automated business processes when collaborating with other parties in order to gear towards the world of semi-automated, complex electronic transactions. Such IT-based coordination of automated business processes was up till the turn of the millennium typically facilitated using Enterprize Data Interchange (EDI) (Bort and Biefeldt, 1994; Kimberley, 1991). However, due to the limited possibilities offered by EDI, its relatively high costs, and its lack of flexibility organizations have begun to move away from EDI towards XML based solutions like web services, which make interoperability between otherwise isolated IT systems more cost effective and manageable.

Additionally, the focus of most business process automation approaches has been on highly stable and well-defined processes. A good example is the workflow oriented research (Workflow Management Coalition, 1995; Georgakopoulos et al., 1995), which has almost solely concentrated on production like systems. Similarly, rigid processes have thus far received most attention in the more recent research on web service based automation of processes. These systems provide a means of defining processes in terms of a set of structured activities that can subsequently be executed and monitored. They have in common that they define a complex model once from which instances are then derived for actual enactment in an automated environment. Such approaches are suitable for standardized intra-organizational and inter-organizational processes. However, as we will see in the next section they are ill-equipped due to their rigid and centralized characteristics to describe and implement the more unstructured and dynamic processes that are inter-organizational processes.

1.1.2 Change

Most research in business process automation and the coordination of automated business processes has focused on the design and execution of fairly rigid business processes and collaborations. However, in the current business environment the assumption of rigidity

underlying this research is no longer a valid one. Nowadays organizations are constantly facing changing business demands for example because of new market conditions, new competitive pressures, new regulatory fiats that demand compliance, and new competitive threats. All of these situations and more drive the business need for the IT infrastructure of an organization to respond quickly in support of new business models and requirements. Additionally, the business models and requirements of an organization are influenced by technological developments, which have the potential to create and support new business activities or make existing activities obsolete. In light of this necessity for change the building and managing of business collaborations becomes a challenge as it requires the dynamic linking the elements of individual organizations together into a consistent and cohesive whole whilst ensuring that the resulting collaborations are and remain conform business requirements.

When contrasted against the trend of a more dynamic and volatile business environment, the application of traditional, rigid business collaboration solutions is not suitable anymore as they are unable to accommodate the constant demand for change as this forces organizations to rapidly respond and adapt their business processes. Specifically, such solutions have two main disadvantages: firstly, the definition and development of the required complex process and collaboration models is difficult and as such often reserved for business modeling specialists. This causes business collaborations to be non-transparent for business people. As such, it is difficult for business people to assess the feasibility and impact of changes on business collaborations. Moreover, it makes that business process and collaboration model creation is a time-consuming and thus costly affair. Secondly, complex models involving many detailed specifications on the exact behavior of business processes and collaborations tend to be rather rigid and as such not very easy to adapt and manage in case of changes in the business collaboration or in the overall business environment.

The observation that the need for change causes major problems in business process automation and collaborations coordinating automated processes is not new, as observed e.g. in relation to workflow systems in (Joeris and Herzog, 1999) and (Liu and Pu, 1997). To tackle this problem many research has focused on comprehending the reasons of change and the requirements it places on the capabilities of business collaborations to cope with this change. The result of this research is summarized in the classification of change shown in Fig. 1.2.

As the figure shows there are four types of change of relevance for business collaboration, being *flexibility*, *formal adaptability*, *dynamism*, and *undefined adaptability*. We collectively refer to these types of change as *dynamicity*. Flexibility is the ability to handle changes that are known at design time, and are known to occur at some specific point during runtime. Flexibility thus places requirements on the modeling of business collaborations, e.g. the capability to abstractly specify what service to use to facilitate dynamic selection at runtime. (Sadiq and Orlowska, 2000) refers to this type of dynamicity as flexibility as well, while (Joeris and Herzog, 1999) refers to it as a-priori flexibility and (van der Aalst et al., 1999b) as structural change.

Formal adaptability is focused on changes that are known at design time yet are unpredictable in nature at runtime. This sounds as a contradiction at first, however, what is

	Design time	Runtime
Unpredictable	Formal Adaptability	Undefined Adaptability
Predictable	Flexibility	Dynamism

Figure 1.2: Change In Business Collaboration

meant by 'unpredictable' here is that these are changes which are known but whose time of occurrence can not be predicted. Such changes must be taken into account when modeling business collaborations. Examples of formal adaptability changes are handlers to take care of exceptions like the timeout of a request. (Sadiq and Orlowska, 2000) calls the capability to handle such changes adaptability, while (Joeris and Herzog, 1999) considers it to be part of its notion of a-posterior flexibility. A-posteriori flexibility itself encompasses the occurrence of unpredictable events and erroneous situations as described in (Han et al., 1998).

Dynamism represents the ability to modify business collaborations at runtime, and specifically to transform these running business collaborations from the old to the new specification when the underlying design is changed. The insertion of new tasks and deletion of existing tasks are standard examples of dynamism. However, dynamism changes can also relate to other matters such as the inputs required for a task, the security requirements for an operation, usage of another service if the selected one is unavailable, and etceteras. (Sadiq and Orlowska, 2000) also uses the term 'dynamism'to denote this type of change, whereas (van der Aalst et al., 1999b) calls them ad-hoc changes. Finally, undefined adaptability concerns changes that are unknown at design time and occur unpredictably at runtime. Undefined adaptability is typically included with formal adaptability in the literature. However, there is an important difference being that undefined adaptability expresses the ability to handle unforeseen changes that occur during execution, whereas formal adaptability is concerned solely with dealing with foreseeable changes. For example,

the receipt of a previously unknown type of message constitutes an unforeseen event.

Now, if business collaboration is to be conducted in a more dynamic manner, then these four forms of dynamicity must be facilitated. Support for flexibility and formal adaptability is needed so organizations can easily make changes to their business collaborations and the manner in which they interact with one another through the exchange of messages between their respective IT systems. Such freedom gives them the means to develop and manage designs for business collaborations in an effective and dynamic manner. At the same time dynamism and undefined adaptability must be facilitated in order for organizations to be able to assess the affect of changes to their business collaborations on running cooperations with partners. This will give them insight on the impact of changes on existing collaborations and consequently will aid them when effectuating these changes. A technology that has become more and more popular over recent years for realizing these types of dynamic business collaborations is service-oriented computing (SOC) based middleware. We discuss this technology in the next section.

1.1.3 Service-Oriented Computing

Service-oriented computing arose in the late 1990s as a new paradigm for distributed computing and e-business processing. It evolved from object-oriented and component computing to enable the building of agile networks of collaborating business applications distributed within and across organizational boundaries. Adopting the service-oriented computing paradigm has the potential to bring about reduced programming complexity and costs, lower maintenance costs, faster time-to-market, new revenue streams and improved operational efficiency. Here we give a brief overview of the SOC paradigm, and outline its usability within the context of the realization of business collaborations. For more extensive information (Papazoglou and Georgakopoulos, 2003) provides an excellent starting point of which the following is an excerpt.

Service-oriented computing utilizes services as fundamental elements for developing applications/solutions (Papazoglou and Georgakopoulos, 2003). In general a *service* can be regarded as "a system that supplies something that people need" (Cambridge Learner's Dictionary, 2006). In service-oriented computing the service being offered is software. The idea of software as a service was first applied by application service providers (ASPs). However, the ASP model resulted in monolithic architectures, highly fragile, customer-specific, non-reusable integration of applications based on tight coupling principles. To resolve these issues the service-oriented computing paradigm has expanded the ASP's notion of services. We refer the reader to (Papazoglou and Georgakopoulos, 2003) for a comprehensive historical overview.

Within SOC services are viewed as independent software entities that perform functions. These functions can vary from very simple requests to complete business processes (see (Veryard, 2003) for an insightful classification). A service is discoverable and dynamically bound, self-contained and modular (Meyer, 2000), stresses inter-operability, is loosely coupled, has a network-addressable interface, has a coarse-grained interface, is location-transparent, and is composable (Booth et al., 2004; McGovern et al., 2003; Stevens, 2003;

Yang and Papazoglou, 2002). Due to their platform-independent nature they can be deployed on any hardware, e.g. to desk-top or lap-top computers running Unix and Windows environments, and to hand-held devices like mobile phones, PDAs, and etceteras.

In the context of business collaboration, SOC provides organizations with the technological means to develop business collaborations in a more cost-effective and flexible manner than earlier solutions such as EDI. Due to the ubiquitous nature of services organizations are able to abstractly represent their IT-infrastructure as a pool of available services (independent of specific technologies). As services stress inter-operability this enables organizations to integrate their IT systems in a standardized manner. Moreover, since services are location-transparent organizations are able to build bridges between both their own systems and those of other organizations in the same way. In such loosely coupled inter-organizational business collaborations organizations can more easily switch partners than with EDI, since development costs are much lower and integration is no longer proprietary. As such, SOC based business collaborations will tend to exhibit more dynamicity compared to the rather static EDI based collaborations, will often be more short lived due to the lower integration costs, and comprise the integration of a larger variety of heterogenous IT systems.

In addition, because services are composable, organizations can combine services to create more complex ones in order to support their business processes. This has the advantage that: 1) organizations can achieve alignment between their processes and the IT-infrastructure by making explicit how these processes utilize the provided IT services, where due to their platform independent nature and loosely coupled nature services can be easily be replaced, service implementations changed, etc; and 2) organizations are in the position to realize their business processes in the most optimal manner by selecting and combining the most suitable and economical services via service composition, where these services may be self maintained or offered by other organizations. This also has the benefit that intra-organizational and inter-organizational processes can be modeled and carried out in the same manner. Moreover, if we have some way of configuring these services on an as-needed basis rather than in a pre-defined manner, then this will greatly enhance the dynamicity of business collaborations.

However, the potential of dynamic service composition can only be harnessed if we are capable somehow of ensuring that the (composed) services perform their work in conformance to the higher level business requirements within the restrictions applicable to these services. Moreover, we need to have some means of ensuring that the behavior of the different services in the service composition is and remains consistent in relation to themselves and to the other services in the business collaboration. In addition, such conformance and consistency must also be maintained as changes like discussed in section 1.1.2 occur. This requires a mechanism with which we can dynamically configure services in accordance with (often changing) demands and limitations. In the following section we will set forth the requirements for dynamic service-oriented computing based business collaboration in more detail.

1.2 Motivation, Requirements and Issues

Now that we have sketched the context in which the research described in this dissertation takes place, in this section we shall explain the motivation behind the presented work. We will also discuss the requirements for business collaboration development and management, after which we contrast them against the current work in this area to identify as of yet to-be-resolved issues.

1.2.1 Motivation

Our motivation for the work presented in this dissertation is twofold in nature: academic relevance and practical significance. From a scientific point of view dynamic business collaboration development and management presents us with a very interesting and challenging research subject. The topic covers a complex web of highly interrelated topics including a wide range of economical, business, organizational and technical issues requiring one to to acquire and apply knowledge from a diversity of disciplines. Moreover, to address these issues new technologies such as service-oriented computing are needed providing the opportunity to work at the cutting edge of IT.

From a practical perspective dynamic development and management of business collaborations has become of paramount importance for organizations to survive and thrive. Enterprizes have awoken to a new world, one in which the old ways have become archaic and have been replaced by the rigors of a global and dynamic world. This world is one in which organizations must be able to easily and quickly adapt their business models, and thus where the capacity to change has become a key element for survival. As such, the capacity (or lack thereof) of organizations to conduct their business activities in a highly dynamic manner has become a matter of life and death.

Given these stakes it is surprising that thus far no adequate solutions have been provided neither by academia nor industry. Even when taking the complexity of the issues involved into consideration it is disappointing to find that proposed solutions only attempt to address some issues whilst simply disregarding others. We are convinced that we can make an important contribution to fill this gap by presenting a coherent and cohesive approach in this dissertation, which provides organizations with the means to dynamically develop and manage their business collaborations.

1.2.2 Requirements

In order to realize the vision of dynamic business collaboration development and management organizations need a sophisticated environment which provides them with the means to:

1. Properly capture the requirements of their business collaborations, where organizations must be able to:

(a) Capture their business processes in a consistent manner, i.e. have *validity*; where business and technical requirements are in *alignment*, that is, are consistent. This ensures that as business collaborations are executed (partially) in the IT infrastructure, they exhibit behavior that is in accordance with the business requirements.

(b) Specify their requirements to cooperate in business protocols; where the defined protocols must be in a state of *compatibility* with the organizations' private activities, that is, are consistent.

(c) Stipulate their business agreements encompassing both business and technical conditions; where any such agreements must be compatible with the business and technical protocols adopted by the respective participating organizations, that is, are consistent.

where for their processes, protocols, and agreements organizations must be able to capture: 1) basic requirements, that is, describe their functional characteristics; as well as 2) advanced requirements concerning for example billing and payment, quality, security, transactional semantics, and so on.

2. Define and manage their business processes, protocols and agreements with a high degree of dynamicity, that is, be able to quickly effectuate changes in new as well as existing business collaborations; i.e. support flexibility, formal adaptability, dynamism and undefined adaptability whilst at the same time be able to ensure that these collaborations are in *conformance* with requirements, and that they remain consistent.

1.2.3 Issues

Unfortunately, current business collaboration solutions are too narrowly focused and not capable of addressing the requirements identified in section 1.2.2. As a result it is very difficult to develop and manage business collaborations in a dynamic manner with existing technologies and standards. Specifically, we identify the following issues that are of yet unresolved for development and management respectively:

1. **Development**

   Most works only facilitate partial specification of business collaboration requirements. Solutions like (ebXML Initiative, 2006; ebXML Initiative, 2002), (Business Process Modeling Initiative, 2002; Business Process Modeling Initiative, 2003) and (van der Aalst, 1998; van der Aalst et al., 2003; Georgakopoulos et al., 1995; Workflow Management Coalition, 1995) focus on definition of functional business requirements, whereas others such as (Fensel and Bussler, 2002), (Curbera et al., 2002), (Arkin et al., 2002), (Christensen et al., 2001), (Banerji et al., 2004) deal exclusively with technical requirements. As such, these solutions are not capable of addressing

both high level, business oriented demands and low level, technically related collaboration requirements. Moreover, they do not provide a way of linking these two types of requirement, which is crucial for organizations to ensure that business activities are supported by their IT-infrastructure. In addition, it is often unclear how extra-functional requirements are to be incorporated (or support thereof is very limited). Also, high level notations such as UML based notations (Booch et al., 1998) are typically used in an ad hoc manner while the resulting models lack formal semantics.

There are a number of research works that span both business and technical business collaboration requirements, for example the work in (Bresciani et al., 2004). Others focus on relating business and IT such as done in (Casati et al., 2003) and (Zeng et al., 2003). The problem with these proposals lies in the fact that they do not take all requirements into consideration, are unable to support high level, abstract requirement definition, and/or do not provide clear cut separation between business and technical requirements. Moreover, many approaches fail to provide uniform techniques for capturing private processes, public protocols and made agreements, nor do they provide the means to capture dependencies among them. Exceptions are (Dijkman and Dumas, 2004) and (Traverso et al., 2004), however, these concentrate on technical and business requirements respectively.

More formally oriented proposals include languages based on simple finite-automata, Petri Nets (van der Aalst, 1998; Verbeek and van der Aalst, 2000), process algebras (Bergstra et al., 2001) like (Hoare, 1985), (Bergstra and Klop, 1985) and (Milner, 1990; Milner, 1993). These proposals have in common that they view both private processes and public collaborations as labelled transition systems, where such a system is the set of states that is an abstraction of all the possible states of a concurrent system (Basten, 1998). Two other avenues pursued are logics such as (Rao et al., 2006) and constraint satisfaction (Aiello et al., 2002) (Papazoglou et al., 2002). Although we do not argue against the need for formal underpinnings of business collaboration design, such techniques also currently do not address all requirements. As such, it remains unclear whether and how their application might be achieved in context of business collaboration design.

2. **Management**

When it comes to service composition and business collaboration in general, most work has focused on development without taking their management into too much consideration. Current solutions like in BPEL (Curbera et al., 2002) and those specified by (ebXML Initiative, 2006; ebXML Initiative, 2002) are pre-determined and pre-specified, have narrow applicability and are almost impossible to reuse and manage. The same applies to works from academia like from workflow in (van der Aalst et al., 2003; Bowers et al., 1995; Workflow Management Coalition, 1995), system development in (Bresciani et al., 2004; Traverso et al., 2004) and organization modelling in (Zachman, 1987). However, the assumption that business collaborations will not change is not a feasible one (as argued for in our discussion in section 1.2.2).

This falsity has been recognized by many works, and consequently there have been several attempts at making business collaboration design more dynamic.

Most approaches have turned to the use of rule based development and management in which the definition and modification of designs is guided by rules. Within workflow the works of for example (Deiters et al., 2000), (Reichert and Dadam, 1997) and (Reichert and Dadam, 1998), (Liu and Pu, 1997), (Joeris and Herzog, 1999), (Christophides et al., 2000) and (Han et al., 1998) come to mind, where the idea is to built so-called flexibility points into the workflow at which decisions can be made as to how to proceed. However, these approaches offer only limited flexibility and adaptability by pre-definition of allowed changes at design time. In addition they only allow the flow of execution to be modified. (Casati et al., 2000) and (Zeng et al., 2003) are similar exponents of rule based design, though then for service-oriented computing based business collaboration development.

Further argument against the above approaches stems from the fact that they are at their base already not capable of capturing all business collaboration requirements. As such, they can only provide limited flexibility and adaptability. (Zeng et al., 2003) for example doesn't clearly separate between business and technical requirements, and consequently no distinction is made between business and technical rules. Moreover, the issue of alignment of these two sets of requirements is not addressed. Another issue left unaddressed is that the semantics of rules differ depending on whether they are governing requirements for private or public behavior. (Bajaj et al., 2006) and (Andrieux et al., 2004) specify how to develop policies for web services, and agreements among services, but the focus is on technical requirements only. Also, how consistency is maintained between policies and agreements is of yet unclear.

In conclusion we have found that current business collaboration solutions are simply not capable of facilitating dynamic business collaboration development and management. Firstly, there is a lack of a coherent and cohesive vision on business collaboration development. Consequently there is no solution that is capable of design of collaborations where all different types of requirements (and dependencies between them) can be specified. Secondly, due to this fact proposals for business collaboration design are unlikely to succeed as they can only offer limited support for dynamic development and management. The extensive literature review underlying these conclusions is presented in Chapter 2. For now we continue in the following section where we give an outline of our proposal to provide organizations with the means they need in order to cooperate with others in a dynamic manner.

1.3 Research Proposal

To meet the requirements for business collaboration in section 1.2.2 and thus address the open research issues, we propose a rule based approach for business collaboration

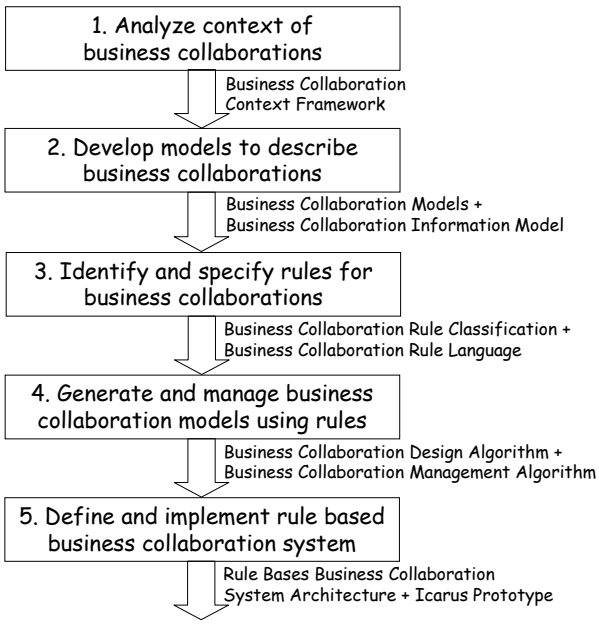

Figure 1.3: Research Proposal

development and management. An overview of the road-map for this approach is shown in Fig. 1.3.

As can be seen in the figure the road-map for our approach consists of five steps. We start in (1) with an analysis of the context in which business collaborations take place. This analysis will result in the definition of the Business Collaboration Context Framework (BCCF). This framework defines a modularization of the context in which business collaboration takes place. Its purpose is to reduce the inherent complexity of business collaboration development and management. Modularization will achieve this as it will allow the development and management process to be sliced into multiple, more manageable chunks. This will enable organizations to focus on specific development and/or management activities whilst at the same time maintain a clear view of their purpose in relation to the overall business collaboration design.

On top of the BCCF we next develop models with which the business collaboration context can be described in (2). The aim of these models is to enable organizations to capture their cooperations with others in the form of models. This will allow organizations to make their business processes, protocols and agreements explicit in an unambiguous and well-defined manner, both from a business and technical perspective. The resulting models can subsequently be utilized for communication and reference purposes. They can also be employed to do simulations as well as constitute the basis for the execution of private processes as well as for the carrying out of cooperative activities. We also specify a generic Business Collaboration Information Model (BCIM) in (2) with which the different

models for business collaboration can be described in an uniform manner.

Subsequently, in (3) and (4) we develop a rule based approach to realize dynamic development and management of business collaborations. The idea behind the approach is to make the business collaboration requirements of organizations explicit in the form of rules, and to then use these rules to drive and constrain the development and management of business collaboration designs. Concretely, design becomes a runtime activity where the business collaboration design shapes itself to its specific circumstances by application of the appropriate rules. As such, business collaboration designs are generated on-the-fly rather than being pre-defined. This will make business collaboration dynamic in two ways: 1) design of business collaborations is governed by explicitly defined and thus manageable rules, which further more can be chained and used for making complex decisions and diagnoses; and 2) business collaborations can be readily changed during design time and runtime by adding new rules and/or re-defining existing rules that handle the change. Simultaneously, rules can also be applied to ensure that the generated business collaboration designs are and remain consistent.

To realize the rule based approach we first make a classification in (3) of the types of rules that are required to handle the identified changes for business collaboration. We also develop the means in (3) to make these different kinds of rules explicit. This will result in the Business Collaboration Rule Language (BCRL), which builds on the BCIM to facilitate rule specification in a generic manner. After that we explain in (4) how the rules defined in BCRL can be utilized by organizations to capture the requirements they have for their business collaborations in an accurate and consistent manner, where for this purpose we develop several mechanisms with which potential anomalies can be detected and remedied. We then show how the resulting rules guide and govern the definition of the different business collaboration models as business collaborations are being carried out. Concretely, this will manifest itself in the Business Collaboration Design Algorithm (BCDA) and Business Collaboration Management Algorithm (BCMA), which will specify in which manner rules can be applied and modified to dynamically design collaborations as they are running while simultaneously ensuring their consistency. Next, we develop a conceptual architecture in (5) for a Rule Based Business Collaboration System (RBCS). We also implement this conceptual architecture in a prototype implementation called Icarus. With this tool we next demonstrate that with Icarus the development and management of business collaborations by capturing their requirements as rules and then use these rules to drive design is practical and feasible. Throughout the different steps we employ formalization and a complex multi-party scenario to showcase the approach's logical consistency and usability respectively.

1.4 Research Goal and Objectives

In line with the sketched research proposal in the previous section the goal of the research presented in this dissertation is stated as follows:

```
The establishing of a rule based approach that enables the
modularized development and management of business
collaborations in a dynamic and consistent manner.
```

Modularized design is achieved if organizations are able to capture the different parts of their business collaborations independent from each other while having the ability to link these parts to a cohesive whole. Dynamicity is realized when organizations can easily incorporate changes in their business collaboration designs, as well as modify their existing business collaboration designs with minimum disruption. Consistency is reached if organizations can develop and manage business collaboration in such a manner that no conflicts between their requirements arise, particularly in light of changes to these requirements. To meet these criteria the following questions must be addressed:

1. **What is the context in which business collaborations take place?**

   We must have a clear understanding in what context business collaborations take place before we can begin thinking about how they can be developed and managed. Specifically, we need to find a way to reduce the complexity that is inherent to the business collaboration context in terms of different types of requirements, dependencies among requirements, and so on.

2. **How do we represent business collaborations?**

   Once we have a complete view of the context of business collaborations we then have to develop an approach with which we can represent and reason about this context. An additional objective in this regard is that the approach should be compatible with existing standards that are currently being applied by organizations in order to facilitate adoption.

3. **How do we make business collaboration design and management dynamic?**

   Business collaborations are not static, rather they are subject to frequent change. Business collaborations must therefore be dynamic both during their design and at runtime. The question is how to achieve this, that is, how will the approach enable the development and management of business collaborations in a dynamic manner? Since most work in the field has limited itself to design of relatively rigid processes, we may have to develop new methods or consider methods from other research domains. In particular, we will look at the possibilities offered by rule based development in this regard.

4. **How can we ensure that business collaborations are and remain conform requirements as well as consistent?**

   The development and management of business collaborations is a complex endeavor, especially in light of the fact that requirements will often be subject to change. This raises the issue of how we can make sure that developed collaborations are and remain conform requirements. Moreover, as business collaborations span intra-organizational

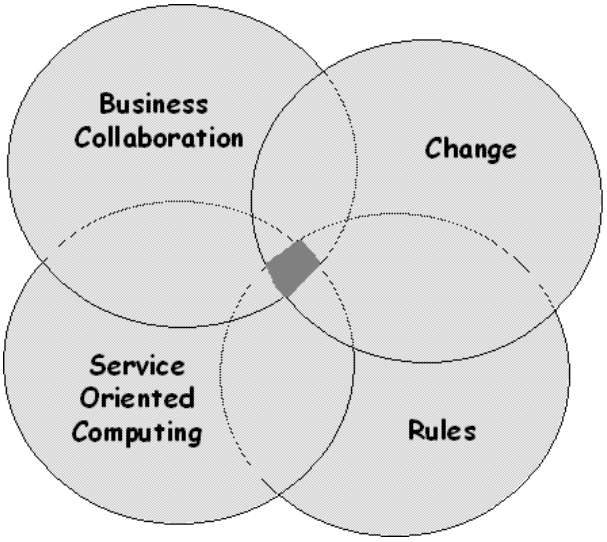

Figure 1.4: Scope Of The Research

and inter-organizational processes across both business and IT, verification of the consistency of designs becomes an absolute necessity for organizations if they are to be able to manage changes to their business collaborations.

In the remainder of this dissertation we will work to achieve these objectives to construct the proposed approach for dynamic business collaboration development and management.

1.5 Research Scope

In order to carry out the research proposal and accomplish the research goal, it is important to properly define the research boundaries particularly as the research encompasses multiple disciplines. Fig.1.4 shows the research boundaries in a Venn diagram.

As the figure illustrates the presented research is at the heart of the intersection of the domains of business collaboration, change, service-oriented computing and rules. In this niche, as we will see in Chapter 2, there has been relatively few work done as presented in this dissertation, and this is thus where the main contribution of the presented research lies. However, in order to obtain satisfactory answers to our research questions, the scope of the research will necessarily not only encompass the darkened intersection in Fig. 1.4, but will sometimes also extend into the individual research domains. In this regard it is important to note that we impose the following limitations per research domain:

- **Business collaboration**

  The research is aimed at the dynamic development and management of business collaborations. An important scope restriction in this regard is that we only consider business collaborations that require such dynamic development and management, i.e. we do not consider static business collaborations. Additionally, we place the following restrictions:

  1. Within business collaboration we concentrate primarily on facilitating the specification and management of the basic requirements of business collaborations. That is, we are mainly concerned with enabling organizations to capture the essence of how they conduct their private and public behaviors from business and technical point of view, and dependencies between them. As such, advanced features such as with regard to payment, transactions, security, quality, assessment, legal issues, and etceteras are mostly excluded. This does not mean that we underestimate the importance of support for capturing these features. However, the magnitude of such endeavor makes it impossible to cover within this dissertation. As a compromise we will give brief explanations as to how the specification of quality and security requirements is accommodated for.
  2. We do not address the issues surrounding semantics within business collaboration. Since the usage of standard terminology is a key necessity in electronic communication, in our research we make the assumption that such terminology has been defined for example by industry standardization bodies (or alternatively that such terminology has been agreed upon by and is shared among organizations). We do envision a role for semantic based technologies like semantic web within business collaboration development and management, since such technologies will be vital to help organization recognize and reconcile differences in the meaning and syntax of the language they employ to express themselves. It is beyond the scope of this dissertation however to take such matters into consideration.

- **Change**

  When it comes to change in business collaboration, the scope of the research is limited to the impact that change has on the development and management of collaborations. As such, we exclude the following:

  1. We argue for the presence of change in the current business environment (and thus in business collaboration) and the necessity to deal with such change. This argument, presented in section 1.1.2, is based in part on arguments found in related literature and in part on common sense. It is explicitly not within the scope of this dissertation to attain and provide statistical evidence for this argument.

2. In this dissertation we only take the effects of change on business collaboration designs into account. We do not consider the reasons of why changes occur, which people are responsible for initiating change, the financial impacts of change, and other related change management issues. Although the relevance of such issues is self-evident, they are outside the scope of this dissertation.

- **Service-Oriented Computing**

  The role of service-oriented computing within the research is limited to its facilitating and enabling qualities with regard to providing an abstraction mechanism for the IT-infrastructure of organizations; where concretely we are interested in how the services provided by this infrastructure can be dynamically configured conform shifting higher level business requirements. In this regard we adopt the following restrictions:

  1. Service-oriented computing comprises a vast domain of related yet widely dispersed issues. Here we focus on the core principles underlying SOC and their applicability in the context of dynamic business collaboration. As a consequence many topics are not taken into consideration in this dissertation. These include automated service discovery, service management, service monitoring, service security, service quality, service design, as well as others. A comprehensive overview of SOC and the multitude of areas it encompasses, is provided in (Papazoglou, 2006).
  2. Web services are a set of technologies that are currently a popular mechanism for implementation of the service-oriented computing paradigm. In the context of our work we are interested in these technologies to the extent that they provide a source of inspiration for how we can represent the IT-infrastructures of organizations in a service-oriented manner. It is not in the scope of this dissertation to discuss developed web service standards and specifications in detail nor do we wish to add new standards to the current web service stack. Interested readers are referred to (Papazoglou, 2006) for a thorough overview.

- **Rules**

  The application of rules within the research is concentrated on their usage within the context of dynamic business collaboration. Specifically, we focus on how rules can be employed to drive and constrain the development and management of collaborations. The research scope is defined accordingly excluding matters such as:

  1. We discuss how rules and rule based approaches can be applied to make business collaboration development and management more efficient and dynamic. It is beyond the boundaries of this research to conduct statistical analysis, which demonstrate in a quantitative manner that rule based approaches possess these qualities.

2. We do not give a full depth overview and analysis of all possible rule representations in this dissertation, in particular with regard to the multitude of existing formal logics. Rather, we focus on those representations that exhibit the properties required in the context of business collaboration.
3. With regard to the definition of formal proofs, we do not develop such proofs for the logics that we adopt unless this contributes to the conciseness and exactness of rule application in business collaboration design. As such, we build on existing work in this area rather than contributing to it. This domain is left to the reader to explore independently.
4. The promise of rules to deliver dynamicity has received widespread attention in industry. This has resulted in a plethora of rule products and tools. Discussion and comparison of these products and tools beyond what is necessary to find useful ideas in relation to the development of a rule based business collaboration system exceeds the scope of this research. Interested readers will find (Chisholm, 2004) and (d'Hondt, 2005) to be interesting points of departure.

1.6 Research Questions

Even within the self-imposed limitations of the just sketched research boundaries, the definition of a rule based approach for dynamic business collaboration development and management requires extensive knowledge of a wide variety of topics. Therefore, we have formulated the following research questions to acquire this knowledge:

1. What is the current state of the art in business collaboration development and management? An analysis will reveal to us what has been done so far in this area. The results of such analysis can influence our original perception of this research domain, and provide us with interesting ideas and tested solutions. Moreover, it will help us to establish the requirements for our research.

2. Based on the analysis of related work and exploration of the business collaboration context, what is business collaboration and in what context do collaborations take place? This question touches upon the core of our research, since we intend to develop a methodology for their development and management. A thorough investigation into this matter will help us gain a clear understanding of how business collaborations can and should be developed and managed.

3. When we have established the context of business collaborations, how can business collaborations then be best represented? We need to get an accurate comprehension of how design is to take place if we are to enable organizations to develop and manage their business collaborations in a comprehensive, cohesive and systematic manner.

4. Once we are in the position to design business collaborations, we next need to investigate how a rule based approach can address the stipulated requirements for

dynamicity. What are the rules that are needed to drive development and management of business collaboration in such a way that the different changes can be accommodated? How do these different rules impact on design and execution of business collaborations? In order to make development and management dynamic we intend to apply rules to drive and constrain it. However, before we can do that we must first know what rules are of relevance, how they are related, when they are of relevance, and so on. After obtaining this knowledge we must to develop a language with which the rules can be made explicit. This language not only has to be expressive enough to capture the business collaboration requirements, but also in such manner that these requirements are manageable.

5. When we have developed the means to capture business collaborations in a rule based form, how do we then ensure that these rules capture the requirements organizations have in such manner that they are complete, consistent and correct? That is, how can organizations define their rules in an accurate manner? Also, how do we apply the resulting rules? How can we verify that business collaborations are conform its rules? How can we ensure that the resulting designs are consistent? In what order are the rules applied? These are questions that we must answer when developing a rule based approach for dynamic business collaboration.

6. As soon as we have determined how to administer rules, the question then becomes as to how changes to rules and policies can be managed? Moreover, how can such a change management system be implemented? Rules and policies will change over time and thus their evolution must be managed. Moreover, the effect of changes to rules on existing business collaboration designs must be managed. Otherwise we will simply move the problem from manually making changes to business collaboration designs to manually modifying rules and policies. To address this issue we need to come up with a way to assess and evaluate the impact of rule modifications on business collaborations.

7. Finally, are rules an integrated part of business collaborations or are they separated into components/services? Should rules be centrally managed or should they be decentralized? How can we design a rule engine to drive and govern business collaboration development and management? In short, how can we implement a rule based approach for business collaboration development and management?

1.7 Research Methodology

In order to find and formulate answers for the identified research questions in a systematic manner we have adopted a research methodology, which provides us with a specific procedure for as to how to accomplish our goal of a rule based approach for dynamic business collaboration development and management. In general numerous paths can be taken when doing research in the domain of information systems. In (Weigand, 2004) a

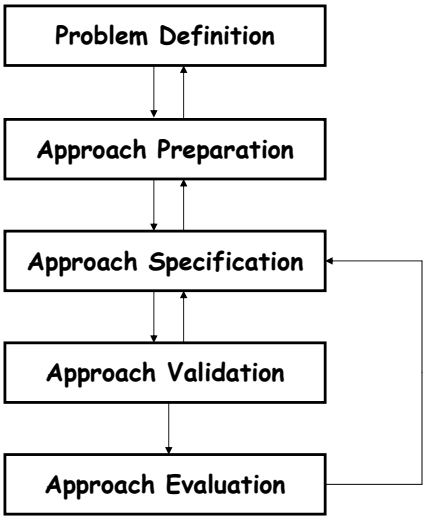

Figure 1.5: Research Methodology

distinction is made between theory and practice oriented research. The first is concerned with developing and/or testing theory, whereas the second is focused on solving a practical problem. Our research is theory oriented as we aim to fill the gap left by current business collaboration development and management solutions with regard to their lack of support for dynamicity. (Weigand, 2004) also differentiates between design (or non-empirical) and empirical research. Non-empirical research focuses on determining whether something is possible and if so how, while empirical research deals with establishing whether or not something is true. We wish to design and implement a rule based approach for dynamic development and management of business collaborations and as such our work is design oriented in nature. To this extent we adopt a six-phased research methodology (as illustrated in Fig. 1.5):

1. Problem Definition

In the first phase, problem definition, we explore the problem domain of the research in order to obtain a clear picture of the difficulties, which are currently present and have yet to be addressed. It is important in this phase that the problem is well-defined to avoid confusion later on, resulting for example in solving the wrong problem. It also is necessary to ensure that the research can be repeated. As already mentioned, we aim to resolve the problem of how to develop and manage business collaborations in a dynamic manner.

2. Approach Preparation

After completion of the problem definition phase we enter the stage of approach preparation. In this phase we identify the requirements for dynamic business collaboration. We analyze the work that has been done in the area of business collaboration development and management to identify interesting solutions as well as spot unresolved problems. This includes both generic approaches as well as those specifically aimed at dynamic development and management of business collaborations. The result will be a set of requirements for dynamic business collaboration.

3. Approach Specification

Once we have clearly established the requirements for the rule based approach, we enter the approach specification phase. In this phase we first define a framework for the context in which business collaborations take place to have a clear idea of what types of business collaboration requirement will need to be covered. We then establish a model based approach to capture this context with in order to assess what the requirements are for the design of business collaborations. When we have the means to describe business collaborations, we develop a rule based approach for developing and managing designs of business collaborations in a dynamic manner whilst ensuring the validity, alignment and compatibility of these designs.

4. Approach Validation

The validation of the presented approach is a threefold endeavor: firstly, to demonstrate its logical consistency we define formal underpinnings during approach specification for the proposed approach. Secondly, to demonstrate its usability we also apply the approach in a complex business scenario in the form of a case study in this phase. Thirdly, in the approach validation phase we show the implementability of the suggested solution by implementing it in the form of a prototype. This threefold validation will substantiate the claim that the proposed approach enables dynamic development and management of business collaborations whilst maintaining their consistency.

5. Approach Evaluation

In the last phase, approach evaluation, we will evaluate the results of our research. The purpose of this evaluation is to determine the merits of our research with regard to the facilitation of dynamic business collaboration development and management in a valid, aligned and compatible manner. Specifically, we will assess to what extent the obtained results address the overall goal of our research as well as the research questions derived from this goal.

In these six phases we employ several research techniques. An excellent overview of research techniques for information systems research is provided in (van den Heuvel, 2002), in which a distinction is made between empirical and non-empirical techniques. Empirical

Research Phase	Research Technique(s)
Problem Definition	Desktop Research, Literature Review
Approach Preparation	Desktop Research, Literature Review
Approach Specification	Method Engineering, Meta-Modelling
Approach Validation	Formalization, Case Study, Prototype
Approach Evaluation	Criteria Definition, Result Assessment

Figure 1.6: Research Techniques

techniques make the assumption that the 'real world' is observable and can be described explained and predicted in terms of its behavior. As such, any ideas that are submitted with regard to this 'real world' must be tested prior to their admittance in a theory. Examples of empirical techniques mentioned include case studies, field study, surveys, field experiments and laboratory experiments. Non-empirical research techniques do not seek validation of ideas before introducing them in a theory. These techniques encompass forecasting, simulation and role playing, subjective/argumentative, and descriptive/interpretive reasoning. Additionally, the technique of engineering is very relevant to our research. Engineering is concerned not only with providing a theoretical basis for a theory, but it is also aimed at the design and realization of technologies. Similar to the choice that has been made in (van den Heuvel, 2002) we view engineering as an individual research technique, combining aspects from both the empirical and non-empirical research technique realms. In line with the type of research that we conduct, we employ mostly techniques from the non-empirical realm. However, for validation purposes we also utilize several empirical techniques. An overview is shown in Fig. 1.6, which gives a listing of the techniques that we employ in the different research phases.

As can be seen in the figure in the first two phases, problem definition and approach preparation, we conduct desktop research and review of the literature. Then, in the approach specification phase we develop our solution to dynamic business collaboration with the help of method engineering and meta-modeling. Method engineering is an engineering discipline to design, construct and adapt methods, techniques and tools for the develop-

ment of information systems (Brinkkemper, 1995). We use method engineering to develop various methods into a cohesive and consistent approach for dynamic business collaboration. During method engineering we employ meta-modeling to help with the integration of these different methods. Meta-modeling deals with the creation of meta-models, i.e. conceptual models that are recursively defined in terms of themselves. In the approach specification phase the need for such meta-models is concerned with the definition of the methods and particularly their interrelationships. Concretely, meta-modeling is used to: 1) represent the different models required for business collaboration design in an uniform manner; 2) capture the rules required for rule based development and management of business collaborations in singular terms; and 3) define how the rules can be used to create and verify models.

As part of the validation of the logical consistency of the approach we apply formalization to these models, rules and the relation between them. Also, throughout the development of the rule based approach we apply the introduced ideas in a theoretical case study based on the AGFIL scenario (Grefen et al., 2000) briefly outlined already in Fig. 1.1 to demonstrate its usability (described in full in Appendix A). This case study describes a complex real life business collaboration between several parties across Europe. Although this case study lacks the empirical nature of a field experiment, we feel it is sufficiently complex to highlight the workings and benefits of the presented research. A second justification for the choice for a theoretical case study is based on the fact that due the inherent complexity of business collaborations and consequently of the presented approach, its application in a real life setting was not feasible given the available resources (as this entails the identification, analysis and specification of all requirements an organization has with regard to a business collaboration, something which can not be done by a mere individual).

We also develop a prototype during approach validation to test the implementability of the proposed approach. The prototype only implements the main functionalities required for dynamic business collaboration development and management omitting additional features. Moreover, no special consideration is given to issues such as performance and security. Rather, the prototype is intended as a working proof of concept, not as a full fleshed rule based business collaboration development and management system. The prototype is explicitly not intended to be evaluated based on criteria such as efficiency, expressiveness, performance, and etceteras. That is, the prototype is strictly used to experiment with the approach with regard to the development and management of business collaboration designs. To be precise, the prototype serves to help with the dynamic creation and modification of the design of the business collaboration described in the case study to showcase the approach's usability. Finally, the prototype is meant to facilitate communication of the research results as well as encourage other researchers to validate these results, or apply the developed approach in field experiments and case studies, to reinforce both the tool and the approach.

Lastly, in the approach evaluation phase we assess the achieved research result. Concretely, based on the set of research requirements identified in section 1.2 and in relation to the stipulated research goal, research objectives and research questions we define a set of criteria against which we will evaluate the result of our research. These criteria are

qualitative in nature yet are measurable in the sense that the research result can be evaluated by examining its support for the individual criteria. Having said that, we adopt the following criteria in this dissertation for dynamic business collaboration development and management:

- **Extensiveness**

  Is the proposed solution extensive enough for organizations to capture the intricacies of their business processes, protocols, agreements and the interplay between them? Concretely, does the solution enable developers to:

  - Describe the different parts of business processes, protocols and agreements as well as the dependencies between these parts using models?
  - Describe business processes, protocols and agreements from both a business and technical perspective as well as dependencies between these perspectives using models?
  - Describe business processes, protocols and agreements as well as dependencies between these processes, protocols and agreements using models?
  - Describe the advanced requirements of business processes, protocols and agreements (regarding e.g. quality and security) from both a business and technical perspective as well as dependencies between these requirements at those perspectives using models?

- **Manageability**

  Does the proposed solution offer adequate support for the manageability of business collaborations in light of changes to an organization's business processes, protocols and/or agreements? Specifically, does the solution enable developers to:

  - Make changes in any part of a business collaboration, that is, modify definitions of business processes, protocols and agreements that were modeled from both a business and technical perspective?
  - Make highly localized, fine-grained changes to business processes, protocols and agreements such as modifying individual quality of service parameters?
  - Make structural, coarse-grained changes to business processes, protocols and agreements like inserting a new activity or replacing a service provider?
  - Assess the impact of any type of changes to business processes, protocols and agreements on other parts of the business collaboration as well as the (possible) effect of these changes on other business collaborations?
  - Keep track of changes to business processes, protocols and agreements for example with regard to the time and reason of change?
  - Create multiple variants of business processes, protocols and agreements to allow the adoption of specialized models for varying circumstances?

  - Apply changes to both new and existing (i.e. already running) business processes, protocols and agreements?

- **Verifiability**

  Does the proposed solution offer sufficient support for verifying and ensuring the accuracy of developed and modified business processes, protocols and agreements? Particularly, does the solution enable developers to:

  - Check and maintain the conformance of developed and modified business processes, protocols and agreements to an organization's requirements?
  - Check and maintain the validity of developed and modified business processes, protocols and agreements, that is, whether these models are complete, correct and consistent?
  - Check and maintain the alignment of developed and modified business processes, protocols and agreements, that is, whether the mappings between a business and technical process, a business and technical protocol or a business and technical agreement are complete, correct and consistent?
  - Check and maintain the compatibility of developed and modified business processes, protocols and agreements, that is, whether the mappings between a process and protocol, and protocol and agreement are complete, correct and consistent (both from a business and technical perspective)?

- **Usability**

  Is the proposed solution easy to use for the development and management of business collaborations? More detailed, does the solution enable developers to:

  - Define models for the business processes, protocols and agreements of organizations in intuitive and familiar terms? That is, are the syntax and semantics of the different models such that they resemble those of existing specifications and standards?
  - Make changes to business processes, protocols and agreements in a simple and effective manner? Moreover, is the manner in which changes can be realized intuitive and uniform for the different kinds of change?
  - Analyze the impact of changes in one part of a business collaboration on its other parts in a (semi-)automated fashion in order to ease the developers' burden?
  - Reuse existing business processes, protocols and agreements in order to increase the speed of development for business collaborations (rather than having to start from scratch each time)?

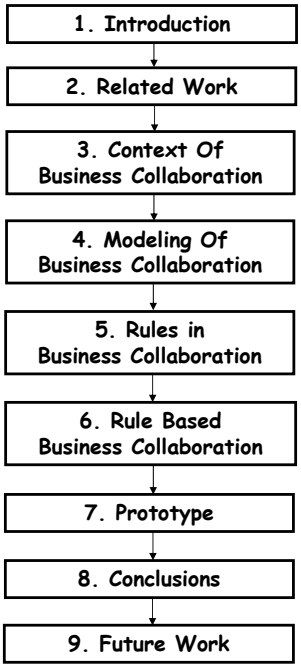

Figure 1.7: Dissertation Outline

Note that many other criteria for our research could be taken into account, most notably concerning the performance and efficiency of developers when defining and managing business collaborations. Although we acknowledge that such, more quantitative, criteria are of interest and would definitely strengthen the intuitive benefits of our research, we do not consider them here as these would require resources for their proper evaluation (both in terms of time and money) beyond those available for this research. Therefore, the criteria defined above are in line with the main objective of this dissertation; being to create a feasible and comprehensive solution for the development and management of business collaborations that helps organizations to do so in a dynamic as well as consistent manner.

1.8 Dissertation Outline

In this dissertation we propose a rule based approach that facilitates the development and management of business collaborations in a dynamic manner. This approach will encompass ideas from a wide range of areas, including but not limited to service-oriented computing, business collaboration modeling and rule driven system development. During development of this approach we will follow the research methodology established in section 1.7. Accordingly, the dissertation is structured as visualized in Fig. 1.7.

As the figure shows we start in Chapter 1 by providing an introduction to the research. In this chapter we report the results of the activities carried out in the problem definition phase. We introduce the reader to the main topics of the dissertation, identify requirements and issues, outline the research proposal to resolve these issues and set the research goal and objectives, and its scope. Next, in Chapter 2 we analyze related work in this area to identify current shortcomings and problems based upon which we extrapolate the requirements for our approach for dynamic business collaboration. In terms of the research methodology in Fig. 1.5 Chapter 2 is the outcome of the work done in the approach preparation phase.

After Chapter 2 we present the work done in the approach preparation phase, being the development of a rule based approach for business collaboration. We start by providing a definition of the context in which business collaborations take place in Chapter 3. Then, in Chapter 4 we explain how we use a model based approach to capture this context in the form of different business collaboration models. Subsequently, we discuss a rule based approach for business collaboration development and management on top of the developed model based approach. First in Chapter 5 we describe the different types of rules required to drive and constrain business collaboration development and management. We also explain the specification of these different kinds of rules in relation to the presented model based approach. After that we show in Chapter 6 how the identified rules are applied to generate business collaboration designs as they are being carried out whilst ensuring the consistency of these designs. We also demonstrate how defined rules can be changed and how the effects of such changes can be incorporated into existing designs.

As part of the approach validation phase we provide formal underpinnings for the proposed approach throughout Chapter 3 to 6. We also exemplify the usability of the presented ideas by applying them in the context of the AGFIL case study earlier displayed in Fig. 1.1 and fully described in Appendix A. Furthermore, we give details on a prototype implementation in Chapter 7, which we build to showcase the practical feasibility of the suggested rule based approach. Then, in Chapter 8 we evaluate the results of the presented research and subsequently draw conclusions as part of the approach evaluation phase. Concretely, we evaluate the rule based approach for dynamic business collaboration development and management against the defined research criteria. In addition we also outline directions for future research in Chapter 9. Finally, we provide a glossary of the terms that appear in *italics* throughout this dissertation for the reader's convenience in Appendix C.

Chapter 2

Related Work

If I have seen further it is by standing on the shoulders of giants; Isaac Newton

The secret of creativity is knowing how to hide your sources; Albert Einstein

In the previous part we introduced the research presented in this dissertation on dynamic business collaboration through rule based development and management. We provided a short background overview of the research's main topics of business collaboration, change, service-oriented computing and rules in section 1.1. We then motivated the need for our work in section 1.2 after which we sketched the requirements for dynamic business collaboration. Subsequently we briefly discussed several current business collaboration automation solutions and contrasted them against these requirements. As a result we found that current solutions are too narrowly focused and not capable of addressing the requirements. For this reason it is very difficult to develop and manage business collaborations in a dynamic manner with existing technologies and standards.

In this part we present the full literature review that led us to this conclusion to further strengthen the outlined requirements. In this review we will particularly focus on those works that have focused on dynamic development and management of business collaboration designs. In this regard it is important to note that it is not our goal here to exhaustively discuss the work done in this regard. Due to the multi-disciplinary nature of our research this is simply not feasible. Rather, the purpose is to gather insight in what research has been done already, in turn enabling us to identify useful ideas as well as pinpoint shortcomings. To this end we analyze the related literature in context of the research objectives we stipulated in section 1.6, where we investigate if and how current works can help to meet these objectives.

Accordingly, the remainder of this chapter is structured as follows: in section 2.1 we explore the literature on the context in which business collaborations take place. Next we investigate current proposals to capture this context, that is, to describe business collaborations in section 2.2. Subsequently we review approaches for dynamic business collaboration in section 2.3. Following that in section 2.4 we analyze proposed solutions for verifying and maintaining the consistency of business collaborations. We conclude in section 2.5,

where we discuss the literature review and highlight important observations to extrapolate as-of-yet to be resolved issues for dynamic business collaboration.

2.1 Context Of Business Collaboration

As we observed in the opening statement of this dissertation business collaboration is about cooperation between organizations by linking their business processes and exchanging messages. The challenge of such collaborations is how to successfully integrate the intra-organizational processes of individual partners with the inter-organizational collaborative process. The interplay of these different processes has been widely recognized in literature as being of relevance for business collaboration. (Dayal et al., 2001) for example promotes development of a collaborative process framework in their survey on business coordination management. One exponent of such framework is the CrossFlow project (Grefen et al., 2000), which developed support for cross-organizational workflow management with which organizations are able to develop a global process encompassing both their respective private processes and the interactions between them, which can subsequently be enacted via a centralized workflow based management system. Another example is found in (Chen and Hsu, 2001) where the idea is to create a virtual cooperative process across organizations. Once the design is accomplished, this process is chopped into individual pieces that are executed by the respective individual organizations.

However, as (Bussler, 2001) and (Fensel and Bussler, 2002) note there are several disadvantages to an approach that considers intra-organizational and inter-organizational as being part of a single, centralized process. Among others the participating organizations have to agree on how to define both their private processes and public process in one workflow definition. This makes it difficult for them to hide these private processes, something which is desirable for example for competitive reasons. It also makes business collaboration inflexible as any change to a private process affects the whole overall process. Hence (Bussler, 2001) proposes to explicitly separate private and public processes, specifically by suggesting to bind public and private processes implemented as business to business protocols and workflows respectively as well as an approach for collaboration management. The advantage of such separation is that organizations only have to agree on the public process. The implementations of this process at each partner are completely independent and private. The binding between the private processes and the public process thus provides isolation and independence.

The separation advocated by (Bussler, 2001) is also argued for by (Traverso et al., 2004), which proposes a development process in the context of service-oriented computing in which both global (public) and local (private) service requirements are incrementally agreed among partners to construct a business collaboration. We find the same train of thought in (Peltz, 2003) in which an explicit distinction is made between orchestration dealing with private processes on the one hand and choreography concerned with public processes on the other. Several other works take the activities suggested by these works one step further by also making explicit the binding between private and public processes.

We find this most notably and explicitly in (Dijkman and Dumas, 2004) which provides a foundational model for designing (composite) services in which four interrelated viewpoints are identified, being interface behavior, provider behavior, and the aforementioned choreography and orchestration (where interface and provider behavior describe what a partner can do and what it expects other partners do respectively). Another area in which we find these viewpoints is in organization modeling, of which the Archimate project (Jonkers et al., 2003) is an example. In their work on an integrated organization architecture description language they distinguish between internal and external behavior of organizations relating these behaviors explicitly through the usage of interfaces.

Another form of separation found in literature relates to the fact that business collaboration requires (semi-)automated, complex electronic transactions and thus technology in order to support the business activities of the partners and help them reach their business goals. As such, development and management of business collaboration applications involves both business and technical requirements and capabilities, and is therefore much more difficult than simply defining and generating for example web services based implementations. In general, for the development of software applications (Object Modeling Group, 2003a) promotes the usage of layers to conceptually represent these different requirements. They advocate a Model Driven Architecture (MDA) based on the idea of separating the specification of how an application works from the details of the way that this application is implemented. To this end they define three types of model, being Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). CIM is intended to capture how an application is to work independent of any technology. PIM is a computationally oriented representation of CIM that details how to realize CIM in a platform independent. Lastly, PSM constitutes an implementation of a PIM on a specific platform/in a particular environment.

In the context of business collaboration a CIM can be used to define business collaboration in a business oriented manner, where the corresponding PIM and PSM constitute its technical realization. Due to the fact though that such CIM will have to concretely specify how the collaboration is to work, it will be very operational in nature depicting what activities to perform when in what order, and so on. Some works state that process management is only concerned with such operational processes, such as (van der Aalst et al., 2003) in their survey on business process management. However, others like (Akkermans et al., 2000) advocate that usage of operational semantics is not sufficient for business collaboration. They argue that failure to acknowledge that a business model is not about process but about value exchanged between actors leads to poor business decision-making and inadequate business requirements. Thus, strategic considerations must be taken into account as well. (Bresciani et al., 2004) reaches the same conclusion but then from a software engineering point of view, where they mark early requirements identification and analysis as being essential in the development and management of software applications.

The need to incorporate high level strategic requirements in addition to operational business requirements and technical requirements is also recognized in the related area of enterprize architecture. Most notable in this regard is the Zachman Framework (Zachman, 1987), which describes a comprehensive framework for enterprize architecture description.

The framework identifies six perspectives from which to describe an organization system development: 1) planner perspective gives definition of direction and business purpose, i.e. concerns strategy; 2) owner perspective specifies in business terms the nature of the business, including its structure, functions, organization, and so on, i.e. makes explicit operational requirements in a computation independent way; 3) designer perspective captures how technology may be used to support owner perspective, i.e. considers technical requirements in a platform independent manner; 4) builder perspective represents a view of the program listings, database specifications, networks, etc, that constitute a blueprint for designer perspective, i.e. provides information about technical realization in a platform specific manner; 5) subcontractor perspective expresses detailed representations needed to realize the blueprint; and 6) functioning organization perspective depicts how a system is eventually implemented and made part of an organization. Less extensive but kindred in spirit is the earlier described Archimate project (Jonkers et al., 2003) taking business, application and technology layer into consideration.

In addition to the separation of concern regarding development and management of intra-organizational and inter-organizational processes from different perspectives, the literature also contains numerous works dealing with the existence of different viewpoints. The basic idea behind such viewpoints is to divide processes into sub-parts, each of which captures a particular subject. Early work in the domain of software development is found in for example (DeMarco, 1978) and (Atzeni et al., 1999) discussing the usage of data flow diagrams and entity relation diagrams to capture data and functional view respectively. (Scheer, 1992) proposes the ARIS development process. In ARIS five views are identified for software development being function, organizational, data, control and output view. Function view comprises functional model, organization view captures organizational model, data view represents data model, control view expresses the order in which activities are carried out while output view conveys the results of these activities (where these results may comprise information, material goods, or some other form of output altogether).

In relation to enterprize modeling (Liles and Presley, 1996) and (Huff et al., 1998) argue the necessity of using five views in an integrated manner to obtain a comprehensive model of an organization: 1) the business rule (or information) view defining the entities managed by the organization and the rules governing their relationships and interactions, 2) the activity view defining the functions performed by the organization (what is done), 3) the business process view defining a time sequenced set of processes (how it is done), 4) the resource view defining the resources and capabilities managed by the organization, and 5) the organization view defining how the organization organizes itself and the set of constraints and rules governing how it manages itself and its processes. Relevant work has also been done in the field of requirements engineering. For example, the approach in (Nuseibeh et al., 2003) describes a requirements analysis method based on the notion of 'ViewPoints'. The idea is that these ViewPoints capture partial requirements specifications, described and developed using different representation schemes. By then capturing the dependencies among the different ViewPoints a traceability mechanism can be established, which in turn enables the management of such dependencies with regard to their consistency.

More recently in enterprize modeling, in (Jonkers et al., 2003) three viewpoints are considered (referred to as aspects) as part of the aforementioned Archimate project. These viewpoints are the structure, behavior and information aspect. Structure aspect describes static organization encompassing both people and information, behavior aspect comprises both the activities done and the order in which they are done, while information aspect deals with business goals, processed information, and offered products and services. Most comprehensive is the earlier mentioned Zachman Framework (Zachman, 1987) defining a data, function, network, people, time and motivation view. Interesting to note is that both (Jonkers et al., 2003) and (Zachman, 1987) recognize that the semantics of their viewpoints vary depending on the layer of abstraction at which they are applied. For example, a data view at planner perspective will be different from that at builder perspective. Moreover, (Jonkers et al., 2003) combines this with the distinction between intra-organizational and inter-organizational processes.

Adding to the previous, in the literature we also find several other issues that are identified as playing a role within the business collaboration context. One of these is the importance of transactional semantics as argued for in for example (Papazoglou, 2003). Business collaborations require such transactional support in order to orchestrate public activities into cohesive units of work and guarantee consistent and reliable execution. Security is another critical issue that must be addressed, as observed e.g. in (Leune et al., 2004) in relation to service-oriented computing. Businesses will be averse to participating in cooperations that do not take place in a trusted environment, for example to avoid problems concerning denial of actions, unauthorized reading of information, and so on. Payment is a third topic that plays a part in business collaborations. Companies typically do not offer their products and services for free, and thus payment must be facilitated such that problems with regard to for example refutability, refundability and annulability do not occur. Quality requirements are also of relevance for the business collaboration context relating to for example delivery time of a product and availability of a service. (Zeng et al., 2004) mentions this for example in the context of service-oriented computing.

Summarizing, the discussion in this section illustrates that the business collaboration context has been extensively analyzed. The most important result that has come out of the described works is the realization that separation of concern is a key concept for successful development and management of business collaborations. Separation is useful to distinguish between business and technical requirements, intra-organizational and inter-organizational processes, and different viewpoints such as organizational and functional standpoint. Additionally, other requirements such as adequate security, monitorable quality of service, clearly defined legal rights and duties, and so on have been identified as being of importance. What is missing though in our view is the overall picture that brings everything together. Most works emphasize parts of the business collaboration context rather than consider them as a cohesive whole. (Zachman, 1987) for example lacks the distinction between private and public processes. (Dijkman and Dumas, 2004) makes this distinction but fails to take business requirements into consideration. (Jonkers et al., 2003) incorporates both separations of concern, however, it identifies only a limited set of viewpoints in comparison to other works. It also does not cover strategic considerations of

business collaborations.

2.2 Modeling Of Business Collaboration

A clear understanding of the context of business collaborations is not of much use if we do not have a proper mechanism to represent this context. By and large all proposals for business collaboration development rely on the activity of modeling to develop representations of business collaborations. These proposals in one way or another facilitate description of (part(s) of) the business collaboration context. A very well established line of research focuses on capturing operational processes. Perhaps the best known exponents of this research are workflow based approaches. Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal (Workflow Management Coalition, 1995). Similarly, (Georgakopoulos et al., 1995) defines workflow as "a collection of tasks organized to accomplish some business processin additiona workflow defines the order of task invocation or condition(s) under which tasks must be invoked". Workflows are formally defined in workflow process models, which are the workflow equivalent of business process models (called workflow process definitions by (Workflow Management Coalition, 1995)). Workflow process definitions, or models, are in general defined using a workflow specification language. There are three basic categories of such languages conform (Georgakopoulos et al., 1995), (Marshak, 1994) and (Mentzas et al., 2001): communication based, activity based and hybrid languages.

The underlying concepts for communication-based languages have been devised in (Winograd and Flores, 1987). These languages assume that the objective of business process re-engineering is to improve customer satisfaction. Therefore they attempt to model the commitments between the human actors involved in the business process. The latter is done by embedding every action in a workflow in the communication activities conducted by a customer and a performer. Activity based languages for workflow specification take a more classical approach compared to communication based ones, excluding business process objectives from the model. The models of these languages focus on modeling the work that is to be done rather than on the commitments between human actors. The emphasis within activity based languages is on ensuring the proper execution of tasks within the process models to achieve a certain goal. For this purpose the workflow model is most often defined as a graph, which specifies the workflow in the business process. A noteworthy example in this regard is XPDL, short for XML Process Definition Language (Workflow Management Coalition, 2002). Optionally it is possible to combine communication and activity based languages, observed in for example (Mentzas et al., 2001) and (Georgakopoulos et al., 1995). This is the case when the process objectives are compatible with both models, e.g. to satisfy the customer by minimizing the number of workflow tasks and human roles.

Workflow has typically focused on intra-organizational processes assuming a single model and centralized execution engine. More recently, as observed in section 2.1, dis-

tributed workflows have been receiving growing attention. Exponents are the earlier mentioned CrossFlow project (Grefen et al., 2000) and the work in (Chen and Hsu, 2001). Another avenue that has been pursued to resolve workflow's problems in the context of inter-organizational processes has been inspired by the emergence of web service technology. Web services are autonomous applications providing some business functionality that is accessible through its public interface, where access to this interface is based on standard network communication protocols. A key quality of web services is that they can engage other web services in order to perform some business functionality (Yang and Papazoglou, 2002). This process, referred to as web service composition, allows the definition of process models as collections of interacting web services.

The defacto standard that has emerged for web service based process modeling is BPEL4WS (Curbera et al., 2002). BPEL4WS, short for Business Process Execution Language For Web Services, is an XML based language in which a process is viewed as a set of complex business interactions involving multiple parties, where each business interaction is perceived as the exchange of a series of messages between these parties. Each process model consists of four parts: the containers, partners, fault handlers and process section. The containers section defines the data containers that are used by the process to maintain state data and process history during execution. The parties involved in the business process are defined in the partners section. Exception handling behavior is defined in the fault handler section. Lastly, the process section contains the definition of the behavior of the business process.

For the specification of the different sections BPEL4WS relies heavily on Web Service Description Language (WSDL) (Christensen et al., 2001), which is a language with which the functional characteristics of web service interfaces can be defined. To incorporate non-functional requirements in WSDL usage of web service policies has been suggested, resulting in the development of a generic policy language WS-Policy (Bajaj et al., 2006). This language has been applied for example to define a security policy language WS-SecurityPolicy (Della-Libera et al., 2005b), which is itself based on several specifications to depict security requirements such as WS-Security (Atkinson et al., 2002) and Web Services Secure Conversation Language (WS-SecureConversation) (Della-Libera et al., 2005a). A proposal similar to WSDL is Web Service Choreography Interface (WSCI) (Arkin et al., 2002), which supports the specification of the observable behavior of a web service as well as the flow of messages exchanged by the web service when interacting with other web services.

Returning to BPEL4WS, two kinds of business processes are distinguished: abstract and executable processes. Abstract processes describe business interactions by precisely specifying the message exchange behavior of the parties involved without revealing their internal implementation. There is a separation from the public and private parts of the business process. This separation allows businesses to keep their internal business procedures secret. Furthermore, they can easily change private aspects of the process implementation without affecting its public behavior. Executable business processes are similar to their abstract counterparts in the sense that they also provide a specification of the message exchange behavior of the parties involved in the business process. However, in an

executable process the external aspects of the process are not separated from its internal workings. This difference between abstract and executable business processes is expressed solely in the availability of different sets for data handling.

The incorporation of executable and abstract process in a single process is reminiscent of the distributed workflow approaches in (Grefen et al., 2000) and (Chen and Hsu, 2001). As such, BPEL4WS suffers from the same problem being that it is not suitable for capturing inter-organizational processes, but rather only for intra-organizational processes (to what (Peltz, 2003) refers to as orchestration). Web Service Conversation Language (WSCL) (Banerji et al., 2004) and Web Services Coordination Framework (WS-CF) (Little et al., 2005) were conceived to fill this gap. Both proposals specify a language with which the coordination of a set of web services can be specified, i.e. with which they enable definition of web service choreography. On the basis of these specifications others have been layered, e.g. to express transactional semantics like WS-Transaction (Cabrera et al., 2002) or to define agreements among web services such as Web Service Level Agreement (WSLA) (Ludwig et al., 2003).

WSLA provides a generic specification language with which the agreed upon interaction requirements between two web services can be captured in service level agreements. In such agreements the level of service is stipulated. In WSLA the parties in the agreement are defined and the services that they offer are described. SLA parameters are the monitored properties of a service, e.g., response time measured by a particular party. Every SLA parameter is assigned one (QoS) metric, which defines how to measure and/or compute the value of the SLA parameters. An example of a metric is response time. One metric can be used by many SLA parameters, e.g., for different Web Services. WSLA focuses on quality of service though. Other works have sought to remedy this limitation. Web Services Offering Language (WSOL) (Tosic et al., 2003) describes a language with which the capabilities of web services can be specified in a more extensive manner, encompassing not only quality requirements but also access rights and other management characteristics. Also, WSOL is extendible whereas WSLA pre-defines the supported SLA parameters. (Paschke, 2005) proposes another competing specification for service level agreement specification by expressing these requirements in terms of rules in their Rule-Based Service Level Agreement Language (RBSLA).

Besides the work from industry on web services the scientific community has also put effort into solutions for business process modeling using web service based technologies. One proposal is the Web Service Modeling Framework (WSMF) that provides a conceptual model for developing and describing web services and their composition (complex web services) (Fensel and Bussler, 2002). It is characterized by a philosophy based on the principles of maximal de-coupling complemented by a scalable mediation service that enables anybody to speak with everybody in a scalable manner. Consequently it uses a combination of ontologies, goal repositories, web services descriptions and mediators to facilitate a peer to peer approach where anybody can trade and negotiate with whoever he or she chooses. Another solution combines the work done in workflow with web services resulting in the specification of eFlow, a system that supports the specification, enactment, and management of composite e-services, modeled as workflow like processes that are enacted

by a service process engine (Casati et al., 2000). Slightly different, in the Self-Serv architecture (Sheng et al., 2002) web services are declaratively composed and then executed in a dynamic peer-to-peer environment. Service composition here is based on state-charts rather than workflow, where an operations input- and output-parameters and produced events are glued together to define the overall composition.

At the same time as web service based technologies were emerging, a set of more generic business process modeling proposals were also developed. One such solution is the Business Process Modeling Language specification (BPML) (Business Process Modeling Initiative, 2002) with corresponding notation (Business Process Modeling Initiative, 2003), developed by the Business Process Modeling Initiative (BPMI). This XML based language provides an abstract model for expressing business processes and supporting entities. The language can be used for expressing abstract and executable processes, which address aspects of enterprise business processes. These aspects include activities of varying complexity, transactions and their compensation, data management, concurrency, exception handling and operational semantics. BPML suffers from the fact though that it is not rich enough to describe inter-organizational processes due to the lack of among others the explicit notion of partners. Swimming lanes are used to implicitly convey such notion. However, we feel this is insufficient for business collaborations, since this does not allow relevant details about the parties involved (like contact information or reputation) to be captured.

Interesting work has also been done by the ebXML consortium. As part of their ebXML suite for e-Business transactions they developed the Business Process Specification Schema (BPSS) (ebXML Initiative, 2006). This schema provides a standard framework with which business systems may be configured to support the execution of business collaborations consisting of business transactions. In this regard a business transaction is considered to be an atomic unit of work in a trading arrangement between two business partners. The followed protocol is very specialized and very constrained in order to achieve very precise and enforceable transaction semantics. Business transactions are realized through the exchange of business documents between the requesting and the responding party. Collaborations can be both binary or multi-party in nature. In a binary collaboration exactly two business partners are involved. It is expressed as a set of business activities choreographed in a particular manner. Multi-party business collaborations express collaborations between more than two business partners. They consist of multiple binary business collaborations choreographed in a certain way. Additionally, the ebXML consortium provides the Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA) (ebXML Initiative, 2002) with which organizations can model their cooperative capabilities as well as specify negotiated agreements, both of which are defined in the same terms as BPSS.

In the scientific community interest has been more diversified than that of industry. One branch that is worth mentioning has explored the usage of so-called event-driven process chains (EPCs). The emphasis in such approaches is on the definition of the control flow of the business process in terms events and functions. Functions perform some business activity when they are triggered by events. Subsequently, they produce events as they carry out those activities. As such, the control flow is expressed in as a sequence of alternating events and functions. To indicate alternative or parallel paths logical operators are often

used. For example, in (Rittgen, 2000) the operators AND (all paths in parallel, where simultaneous processing is possible), XOR (one alternative path is chosen), and OR (one or several paths). Besides these three operators (Loos and Fettke, 2001) also distinguishes the operator SEQ, which indicates that all paths may be performed in arbitrary order, but not simultaneously. An XML based example of EPCs is the EPC Markup Language (EP-ML) (Mendling and Nttgens, 2004). In addition to the specification of the process control flow other kinds of information may be included in an EPC. Such information usually depicts the use of resources for the carrying out of functions and the interactions with the data structure of the organization. Because of their inherent distributed nature EPCs are suitable for both intra-organizational and inter-organizational processes.

Another interesting research area is that of role and agent based approaches. The motivation behind such approaches is that often in process models it is unclear who is responsible for what, since many of them (such as workflow) tend to have decomposition related to function. However, for an individual (or group) to carry out their activities, they need to know what activities they must take part in, in what order those activities must take place, and what other individuals or groups they must interact with. This is particularly of relevance in inter-organizational processes, which are essentially interactions between multiple individuals and/or groups. An example of a role based approach is (Dubray, 2003), which suggests to specify the foundation of a metamodel of a business process definition in terms of message exchanges between two roles. This enables the definition of decentralized processes which involve business-to-business collaborations as well as user interactions and application-to-application integration in a technology neutral way. (Jennings et al., 1996) describes an agent based approach to business process management in which responsibility for enacting various components of the business process is delegated to a number of autonomous problem solving agents. To enact their role these agents interact and negotiate with other agents in order to coordinate their actions and to buy in the services they require. The possibilities of agent based business collaboration are also explored in (Wagner, 2003), which proposes agent-object based modeling for organizational information systems. Concretely, processes can be defined in terms of interacting agents where the behavior of these agents is governed by action and reaction rules.

A topic that has recently drawn a lot of scientific attention is that of the semantic web. The work done in this area is the successor of earlier data-oriented business process solutions. Central in data-oriented approaches is the idea that information is key to the business process. Such approaches start out by defining the data structure in a process. Subsequently, operations that can be applied to this structure are developed, which constitute the different process tasks. In quick overview their history has passed from data flow diagrams (DFDs) in (DeMarco, 1978) and (Yourdon, 1988), control flow diagrams (CFDs) in (Hatley et al., 2000) and ERDs (Atzeni et al., 1999), to Object-Oriented Modeling (OOM) in (Meyer, 2000) and (Booch et al., 1998), to the more current semantic web based attempts based on ontologies (e.g. defined in RDF (Manola and Miller, 2004) or OWL (McGuinness and van Harmelen, 2004)). The most noteworthy exponent of ontology based business process modeling is DAML-S, which is being developed by the DAML Program (DAML Services Coalition, 2003). DAML-S is a semantic oriented ontology for

describing properties and capabilities of web services. DAML-S aims at providing a service description framework that exploits semantic web resources and their semantics to support reasoning about available services. As part of its framework DAML-S utilizes process models to support the modeling of compositions of services. Other interesting works include (Cardoso and Sheth, 2003) and (Laukkanen and Helin, 2003) both of which focus on service discovery and matching, plus providing a general description of how ontology based service compositions for workflow can be developed.

All solutions discussed so far tend to focus on either business or technical requirements and/or on intra-organizational or inter-organizational processes. Proposals covering both business and technical requirements, and particularly the dependencies between them, are more scarce. One work currently in progress is SOMA, short for service-oriented modeling architecture (IBM, 2006), but has of yet not been made public yet. More concrete is the suggestion in (Veryard, 2003) to express business and IT pervasively in terms of services resulting in business and technical services respectively, which can consequently be mapped. To the best of our knowledge though this idea has not been worked out yet. Such complete approach is offered by Tropos (Bresciani et al., 2004), which covers the entire cycle from early requirements analysis to implementation. A particularly interesting feature is that it explicitly captures goals and high-level requirements, issues that are often not taken into consideration when talking about process modeling and enactment. The same applies to the usage of the different UML (Booch et al., 1998) models in the context of business process automation. Whereas use case diagrams are useful for requirements analysis, others such as class diagram and activity diagram help capture operational level like requirements. The UML diagrams do not provide support for all various requirements though. Also, they typically lack formal semantics making the resulting models sometimes imprecise and ambiguous. Moreover, the constructs used to build the different diagrams relate more to the domain of software development than that of business collaboration, making collaboration development more the domain of IT specialists rather than business people.

(Traverso et al., 2004) is another work that encompasses both business and technical requirements. Its main focus though is on resolution of conflicts between global and local requirements, that is, between intra-organizational and inter-organizational processes. A related work in this regard is presented in (Dijkman and Dumas, 2004) in which dependencies among these two types of processes are identified and made explicit in the context of service-oriented computing, so they can be formally verified. Covering both intra-organizational and inter-organizational processes at both business (operational requirements) and technical level is the Archimate project (Jonkers et al., 2003). Archimate also defines sets of mappings between them to express dependencies. A more abstract approach is described in (Dietz, 2006), which presents an ontology based methodology for enterprize modeling. In this methodology the focus is on the communication between people at different levels of abstraction, and how this communication leads to activities that drive the behavior of the organization. Finally, (Gordijn et al., 2006) offers a value oriented modeling technique for business collaborations, where the emphasis is on capturing the value propositions made between the different organizations in the collaborations. The resulting models are then related to i$*$ models (Yu, 1997), which focus more on the

stake holders involved and their interests and goals. Although the ideas proposed in these different works are interesting, it remains unclear how they can be applied in the context of dynamic business collaboration development and management.

A separate category of process modeling languages is formed by more formally oriented works. These have in common that they are based on the notion of describing a concurrent system as a labelled transition system, where such system system is the set of states that is an abstraction of all the possible states of a concurrent system (Basten, 1998). Transitions between states are depicted by transition relations, where each transition is labelled with an action. An action can be any type of activity performed by a concurrent system, for example the sending of a message. Based on the concept of a labelled transition system a set of processes can be depicted in a process space. This space can be defined as a labelled transition system extended with a termination predicate on states. Each state in the system can be interpreted as the initial state of a process, where this process itself is also viewed as a labelled transition system with a termination predicate. This termination predicate indicates in which state(s) a process can finish successfully. If a process cannot perform any actions and it is not in one of these states, then it is in a deadlock situation.

A first technique based on labelled transition systems are simple finite-automata, i.e. basic state machines, where a state machine can be defined as a device that maintains the state of something at a certain time and can alter this state in reaction to input as well as cause an action or output as a result of a changing state. If we view a process as a complex collection of interrelated states, then we can express its control flow in a finite-automata. We can subsequently simulate its behavior to assess how the process moves from state to state. Petri nets offer another technique, and are a special form of graphs constituting of places, transitions, directed arcs and tokens. Places are connected via directed arcs to transitions and vice versa. Places contain tokes, which may represent signals, events, conditions, and so on. Transitions are fired through the presence of tokens in their in-place(s). As a result the distribution of tokens is changed. When applied to process modeling, Petri nets can be used to check their control flow for reachability, deadlocks, and conflicts. The control flow is expressed as places containing conditions, and transitions representing tasks. Tokens exchanged during execution depict the message (and data) flow of the process. Example works include (van der Aalst, 1998) and (Hamadi and Benatallah, 2003) using Petri Nets for workflow and service composition representation respectively.

Simple finite automate and Petri Nets are usually considered to be suitable for graphical representations of concurrent systems. In contrast, the techniques of process algebras and formal logic are more apt for behavioral analysis of these systems due to their symbolic nature. Process algebra is a formal description technique designed for complex computer systems, especially those involving communicating, concurrently executing components (Bergstra et al., 2001). Many process algebras exist, such as CSS (Milner, 1993), CSP (Hoare, 1985) and ACP (Bergstra and Klop, 1985). An algebra that has received some popularity, specifically in the context of web service based business processes, is π-calculus (Milner, 1993). In π-calculus a process is described in terms of systems that are linked to one another. Each link is a channel along which values may be communicated. This

is very akin to the definition of an process as a collection of interacting services over communication channels. Besides enabling verification and validation a very interesting property of π-calculus processes is that links between systems may be created, moved and deleted, which in theory enables processes to be dynamically created. Formal logics define processes as collections of predicates based upon which reasoning can take place to analyze characteristics of these processes. Examples include situation logic, temporal logic, propositional logic, as well as other forms of logic.

In addition to the above, efforts have gone into the modeling of (among others) security, transaction, quality, payment and legal issues. It would go beyond the scope of the dissertation to discuss all these areas in details here. However, to illustrate, (Nadalin et al., 2006) and (Della-Libera et al., 2005b) propose a way to define security requirements for web services, where the first identifies security properties and the second allows definition of policies based upon these properties. Similarly, transactional models are suggested by works like (Cabrera et al., 2002) to define such semantics for service composition based processes. Quality of service is also receiving increased attention e.g. in works like (Froland and Koistinen, 1998) and (Zeng et al., 2004), which identify and capture quality requirements for distributed object systems, and web services and compositions thereof respectively. Legal semantics have also been taken into consideration, for example in (ebXML Initiative, 2006) which specifies a rudimentary 'legallyBinding' property in its BPSS and in (OASIS Legal XML eContracts Technical Committee, 2006) which will aim at enabling the creation, maintenance, management, exchange, and publication of contracts. In the sphere of web services the proposed WS-Agreement (Andrieux et al., 2004) provides a mechanism to define agreements about web service interactions.

Summarizing, as the discussion in this section has demonstrated the modeling of business collaborations has received widespread attention, ranging from strategic models in (Bresciani et al., 2004) to service composition languages like BPEL4WS (Curbera et al., 2002). The problem in our opinion is that this research area makes a chaotic impression as it contains a plethora of specifications that have been developed without any apparent relation to each other. As such, there is no single cohesive solution with which the entire context of business collaborations can be captured. Exceptions to a large extent are works like (Bresciani et al., 2004) and (Jonkers et al., 2003). However, these do not fully support definition of all requirements specifically those related to advanced areas like quality, security, and etceteras. Works that do facilitate modeling of such requirements tend to focus on one level of abstraction. Because of this they fail to make explicit dependencies among these requirements, e.g. when considering security from a business and technical point of view respectively. Similarly, it is typically not possible to capture the dependencies between the requirements of related private business processes and public business collaborations.

2.3 Dynamicity Of Business Collaboration

Due to the long history of change in business collaboration development and management approaches much research has gone into addressing change, that is, in making the automation of business processes and coordination of business collaborations more dynamic. To recall from section 1.1.2 in Chapter 1, dynamicity comprises four types of change being flexibility, formal adaptability, dynamism, and undefined adaptability. Starting with flexibility, one option that has been explored is to build flexibility into a design. The idea is to deliberately specify an incomplete model, thus supporting a high degree of variation in the final design structure at runtime. This allows actual design instances to determine their own, unique enactment of the underlying model e.g. to accommodate the late binding of resources and selection of alternative execution paths.(Han et al., 1998) distinguishes two popular types of approach to dynamic workflow definition that enable such built-in flexibility: meta-model and open-point approaches.

Meta-model approaches utilize meta-models to offer support for structural changes in the workflow graph, such as the adding, editing and deleting of tasks. However, they often neglect changes to a single task. As such, local adjustments cannot be made in these approaches. An example of a meta-model approach can be found in (Sadiq and Orlowska, 2000) in which a set of business process transformations is defined that enable structural modifications to process models. The work presented in (Christophides et al., 2000) proposes some interesting ideas in workflow inter-operability. It describes an infrastructure to support dynamic aspects in planning, scheduling, and execution by introducing workflow schema templates. Reuse of existing workflow schema and templates can be achieved by schema splicing. In contrast, open-model approaches focus on providing special points in a workflow graph at which modifications may be made. Examples of such modifications can range from the late binding of resources to the dynamic definition of sub-models at runtime. An example of an open-point based solution can be found in (Georgakopoulos et al., 1995). In this solution rules are utilized to govern change in an open-point based approach to dynamic workflow processes. These open points allow a number of predefined changes to be made to the workflow structure, such as the insertion and deletion of tasks. A more recent example of such approach in the context of service-oriented computing is BPEL4WS (Curbera et al., 2002), which supports late binding of abstract roles to concrete web service providers.

Unfortunately, structural changes are not well supported in open-point approaches. Therefore, some have proposed to synthesize the meta-model and open-point approaches. In (Han et al., 1998) the skeleton for such an approach is proposed, which includes dynamic definition of the workflow structure, dynamic binding of resources, local decision-making and exception handling. As such, it theoretically can facilitate both structural and local adjustments in workflow graphs. However, as far as we know this idea has not been further explored within dynamic workflow research. An alternative to meta-model and open-model approaches is offered by solutions in which process models are generated at the moment that they are needed. Typically the generating process is driven by rules. In the area of AI (Aiello et al., 2002) has for example looked at goal-driven development of

service composition via a planning algorithm. (Papazoglou et al., 2002) proposes an XML based request language based upon which service compositions plans can be generated that provide the functionality requested by the user. However, the existence of some pre-defined workflow schema is assumed from which a composition can be created. (Zeng et al., 2003) avoids this problem in their rule inference framework DYflow in which rules are used to drive the development of service compositions. Depending on the specific requirements services are composed on the fly, where decisions concerning their ordering and binding are governed by rules. In a related work (Zeng et al., 2004) discusses how selection of services can be carried out at runtime in context of a specified composition plan. The selection process is driven by quality of service considerations such as response time and reliability, where it is formulated as an optimization problem solvable via usage of efficient linear programming methods. The latter work suffers though from a limited scope as it only takes the service level into consideration as such not considering business requirements.

Taken to the extreme web service composition based approaches attempt to realize automated service composition in which based on a given user objective a workflow of services is automatically composed without any user intervention. (Rao and Su, 2004) provides an excellent overview categorization and discussion of such approaches in the context of the semantic web. Based on situation calculus (McIlraith and Son, 2002) defines a method with which software agents can reason about web services to perform automatic web service discovery, execution, composition and inter-operation. The user request and constraints are expressed in first-order situation calculus (a logical language for reasoning about action and change), whereas services are represented using DAML-S so the agents can reason about them. (Medjahed et al., 2003) uses a rule based approach to create composite services from high level declarative descriptions. The method uses composability rules to determine whether two services are composable. Multiple plans may be generated in which case the user can select the most desired one for example based on cost, quality of service, and security. A work resembling the latter is SWORD (Shankar et al., 2002) in which the service requester defines the desired begin and end state, after which a plan is generated to move from first state to the next using a rule-based expert system. Although these works propose intriguing ideas, it remains to be seen whether automated service composition is feasible. Especially the lack of shared semantics is an issue that will hinder such composition, something which is not an unimportant issue in the context of business collaboration.

A different type of rule employment is suggested by other works in which dynamic requirements are associated with more static knowledge. The idea is that there are models with which business collaborations can be described, which are then annotated with rules to capture requirements that are more prone to change. The OGM for example has developed OCL (Object Modeling Group, 2003b) and its proposal for Semantics of Business Vocabulary and Rules (SBVR) language (Object Modeling Group, 2006) (previously known as Business Semantics of Business Rules). OCL facilitates the specification of constraints to express requirements for the classes in a class model, e.g. to depict how to calculate a gross price out, to govern the linking of class instances at runtime, and etceteras. Two other proposals are Semantic Web Rule Language (Horrocks et al., 2003) and RuleML, which both

aim to facilitate the specification of rules. As such, they may be used to express business collaboration requirements, which then can be managed. However, it is of yet unclear how this may be accomplished in the context of business collaboration as this is not the focus of these works. More concrete work on this matter is SweetRules (Grosof et al., 1999). As part of SWEET, SweetRules is used to express the requirements in e-contracts that can then be used during enactment to drive and constrain contract execution.

Work has also been done with regard to formal adaptability to handle exceptions and errors in business collaborations. One option is to utilize backward recovery where the goal is to bring a business collaboration back to its last known consistent state. The basic principle of backward recovery adaptation therefore is to automatically suspend execution and begin considering adaptation steps in order deal with the exception. Here the business collaboration is suitably modified so that execution can resume in a consistent state, which is conform to the changed situation. Well known exponents of backward recovery are compensational activities in transactions. Although useful in database-like settings, the application of backward-recovery based approaches such as (Joeris and Herzog, 1999) and (Liu et al., 2001) is usually limited in business collaboration because it rather inefficient to undo work (and sometimes even impossible). Moreover, (Tartanoglu et al., 2003) makes the valid point that in the context of service-oriented computing based business collaboration, backward recovery would imply locking of resources as well as presence of pre-defined compensational operations for each operation of a service. Both assumptions are often not realistic in business collaboration development and management.

Due to the shortcomings of backward recovery many works have looked at so-called forward recovery, which conveys another plan of attack (like done in (Eder and Liebhart, 1996), (Curbera et al., 2002) and (Cabrera et al., 2002)): whenever an exception occurs, take care of it via exception handlers (or a similar mechanism) that handle the problem. Here the idea is that execution is continued in a normal fashion via the handler dealing with the exception. (Liu and Pu, 1997) defines such a mechanism for exception handling in their language ActivityFlow. At build time exceptions may be specified as well as any corresponding compensation activities. Additional possibilities are offered at runtime to support error recovery by allowing the definition of user or system recovery routines. A similar idea underlies the fault handler section in BPEL4WS (Curbera et al., 2002). A related suggestion is made in (Li et al., 2003), however there the modeling and handling of exceptions relies on continuations, listeners as exception handlers, and on policies, or strategies, for continuation. These so-called exception handler policies are thus responsible for taking care of any occurring exceptions. (Hagen and Alonso, 2000) proposes a solution highly akin to programming languages like JAVA to implement exception handling in order to realize more reliable processes. The problem with such solutions is that it is neither possible nor desirable to include all possible exceptions in the business collaboration models at design time and as such their level of applicability is limited. Moreover, including many exceptions makes models very complex. (Brambilla et al., 2005) acknowledges this and therefore uses a combination of exception handling and user based intervention to handle unexpected events and erroneous situations.

Attention has also been paid to the modification of running business collaborations to

accommodate dynamism. Essentially there are three routes that can be explored (see also (van der Aalst et al., 1999b) and (Sadiq and Orlowska, 2000)): 1) abort the collaboration and (optionally) restart in accordance with the new design; 2) let the collaboration finish; and 3) transfer or transform the collaboration from the old design to the new design. Examples of transformation include the insertion and deletion of tasks, the rolling back of completed tasks and the execution of compensation activities. These are commonly referred to as dynamic changes (Ellis and Rozenberg, 1995). Cancellation of a business collaboration is possible of course, though it will most likely involve some form of backward recovery. This depends on whether the change is affecting an already completed part of the collaboration. Cancelling may not always be the desired solution as it typically results in loss of money and time in terms of to be undone work. Letting the collaboration finish is another option and does not demand any additional effort. This can be an attractive option unless of course the change has a profound influence on the completed or yet to be performed part of the business collaboration.

The third option, migration, relatively speaking causes the most theoretical and practical problems (van der Aalst et al., 1999b), as it involves changing the collaboration in such a manner that it becomes conform the new requirements without losing consistency. These problems with dynamic changes have their root in the fact that modification of existing collaborations may not only require part of the work to be undone and/or compensated, but also that these collaborations must be modified in such a way that they deal with the change while remaining consistent at the same time. Several works have proposed some form of transformation mechanism to accommodate modifications to processes at runtime. The idea is that there are a number of pre-defined operations one can apply to a process at runtime of which it is (formally) proven that these leave the process in a consistent state. For example, in (Joeris and Herzog, 1999) workflow changes are specified by transformation rules composed of a source schema, a destination schema and of conditions. The workflow system checks for parts of the process that are isomorphic with the source schema and replaces them with the destination schema for all processes for which the conditions are satisfied. The approach described in (Georgakopoulos et al., 2000) allows for automatic process adaptation. It presents a workflow model that contains a placeholder activity, which is an abstract activity replaced at run-time with a concrete activity type. This concrete activity must have the same input and output parameter types as those defined as part of the placeholder. In addition, the model allows to specify a selection policy to indicate which activity should be executed.

Similarly, (Reichert and Dadam, 1997) and (Reichert and Dadam, 1998) provide a formal definition that supports the dynamic changing of running processes. It is based on a formal workflow model, ADEPT, which defines a minimal set of operations to insert, delete, iterate, save and skip tasks in a dynamic fashion. (van der Aalst et al., 1999a) is another exponent of such approach based on inheritance, where the idea is to employ four forms of inheritance to facilitate Petri Net workflow specification as well as a set of inheritance transformation rules. Then, proven formal properties concerning consistency of a super workflow can also be maintained for its sub workflows given that the inheritance transformation rules hold. An alternative route is taken by works like (Geppert and Tombros,

1998) and (Meng et al., 2002), which express allowable modifications at runtime in terms of transformation rules and policies. This is similar to (Casati and Shan, 2001) which promotes usage of so-called consistency rules to govern modifications. Although (Geppert and Tombros, 1998) and (Meng et al., 2002) present interesting ideas their scope of change stays limited to modifying task structure by governing the control flow of processes or the ordering of events in case of (Casati and Shan, 2001). This is too limited for the range of changes that might occur in a business collaboration, encompassing not only control flow changes but also changing inputs/outputs, shifting temporal requirements and dynamic resource selection.

Concluding, we feel that the above described works offer useful insight how to accommodate for change in the context of business collaboration. However, they typically focus on a specific type of change like flexibility or formal adaptability. What is lacking is a generic mechanism in which a wide variety of changes can be handled. In addition, the support that is offered by existing solutions has limited applicability, concentrating for the most part on making changes to the order of activities or to resource/service selections. Other kinds of changes that can occur within the context of a business collaboration are typically not taken into consideration. Moreover, the issue of how to maintain alignment of models describing business and technical requirements is left unaddressed. The same applies to the dependencies existing between private and public processes. Finally, an important topic that we feel has been largely ignored, is the management of requirements (rather than the impact of changing requirements on existing business collaborations). For example, rule based approaches as suggested in (Shankar et al., 2002) and (Zeng et al., 2003) do not provide means to maintain different variants of the same requirements, something which is necessary if organizations are to be able to evolve their policies and regulations to govern new processes whilst retaining the old ones to guide existing business collaborations.

2.4 Validity Of Business Collaboration

Organizations are dependent on their private and public business processes for their existence. It is therefore of paramount importance that these processes are modeled and carried out in a manner conform to requirements whilst preserving consistency. Due to the complexity and size of business collaborations verification of these characteristics poses organizations with a formidable challenge. Moreover, the need to make business collaboration development and management dynamic further complicates matters as organizations must now be able to not only verify their business collaborations for conformance and validity, but also to ensure that they remain that way. In combination this requires mechanisms for the formal verification for business collaborations. The necessity for such mechanisms has been long recognized, and consequently several options have been pursued.

A first area of related research has looked at model checking. The idea behind model checking is that we have a design (called a model) and a property (called a specification) that the design is expected to satisfy. Using a model-checking tool it can then be determined whether the given model satisfies given specifications. If not, the tool generates a

counterexample which details why the model doesn't satisfy the specification. By studying the counterexample the problem(s) can then be pinpointed and corrected. The idea is that by ensuring that the model satisfies enough properties, the confidence in the correctness of the model is increased. In the context of process modeling the model is usually expressed as a transition system, i.e directed graph consisting of nodes (or vertices) and edges. A set of atomic propositions is associated with each node. The nodes represent states of a process, the edges represent possible executions which alters the state, while the atomic propositions represent the basic properties that hold at a point of execution.

One set of solutions that has been proposed to check control flow related properties is based on the idea of state machines, particularly finite state machines (FSMs). A FSM or finite automaton is a model of behavior composed of states, transitions and actions. A state stores information about the past, i.e. it reflects the changes in state from the beginning of the process to the present. A transition indicates a state change and is described by a condition that needs to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment. The properties of FSMs that can be analyzed include deadlock and reachability. Also, optimization of a process by reducing the number of needed steps can be performed. Several extensions of basic finite automata have been suggested, the most noteworthy one being Turing Machines. For more information on finite state machines, we refer the reader to (Wagner, 2006). An example of an automata-based model checker is SPIN (Holzmann, 2003). Models to be verified are described in Promela (Process Meta Language), which supports modeling of asynchronous distributed algorithms as non-deterministic automata. Properties to be verified are then expressed as Linear Temporal Logic (LTL) formulaes. These are subsequently negated and converted into so-called Büchi automata after which they are tested.

A more complex proposal for control flow verification is based on the idea of Petri Nets. A Petri Net graphically depicts the structure of a distributed system as a directed graph. Such graph consists of places, transitions and directed arcs. Places may contain any number of tokens. Transitions represent the move from one place to another, where places and transitions are connected via directed arcs. The places from which an arc run to a transition are called the input places of the transition. The places to which arcs run from a transition are called the output places of the transition. When the input tokens required by a transition become available, it fires. When a transition fires, it consumes the tokens from its input places, performs some processing task, and places a specified number of tokens into each of its output places. Petri Net graphs can be formally verified for several properties, most notably reachability (whether a state can be reached), liveness (whether the Petri Net can lock up) and boundedness (whether places adhere to the maximum amount of tokens specified). Applied to business process verification a place in a Petri Net represents a process state. Transitions then govern how the process moves from one state to another. They constitute activities taking tokens as input and producing tokens as output. The work in (van der Aalst, 1998) is a classical example of such approach applied in relation to workflow. (Verbeek and van der Aalst, 2000) provides an implementation of that approach with the Woflan prototype, which can be used as a workflow diagnosis tool. In web service composition Petri Nets have also been applied for example in (Hamadi

and Benatallah, 2003). More complex Petri Nets are also in usage such as coloured Petri Nets, timed Petri Net and prioritized Petri Nets. We refer to (van der Aalst, 1998) for an overview.

An alternative that has been suggested to Petri Nets is the so-called Actor Model (Hewitt et al., 1973). A major motivation for the development of this model was that Petri Nets can capture control flow, but not data flow. The Actor model is based on the idea that everything is an Actor. An Actor is considered to be a computational entity capable of receiving a message and in response can concurrently send new messages to other Actor(s), create new Actor(s) and/or determine how the next incoming message will be received. All interaction between actors is done via message passing, where such messaging is asynchronous in nature and there are no restrictions imposed on message arrival order. In the context of business collaboration the Actor Model provides intuitively appealing features to describe how parties within a collaboration interact. Each party becomes an Actor where their interactions are modeled in terms of asynchronous message passing. One useful feature of the Agent Model is that it allows to verify whether two parties exhibit the same behavior in terms of their message passing capabilities. Another important characteristic of the Actor model is that it is composable. That is, more complex models can be formed out of more basic ones. This enables the construction of highly complex Actor Models while retaining the ability to reason about these models.

Another group of model checking based validation approaches are found in the family of process algebras. Process algebras (or calculi) provide a tool for the high-level description of interactions, communications, and synchronization between a collection of independent processes. Examples of such calculi include CSS (Milner, 1990), CSP (Hoare, 1985), ACP (Bergstra and Klop, 1985) and π-calculus (Milner, 1993), where the latter underlies the earlier discussed BPML (Business Process Modeling Initiative, 2002) and BPEL4WS (Curbera et al., 2002). These calculi have in common that they enable representation of interactions between independent processes as communication (message-passing) rather than as the modification of shared variables. Processes are themselves described using a small number of primitives and operators for combining those primitives. The operators are then defined using algebraic laws, which allow process expressions to be manipulated using equational reasoning. Like Actor Models, processes expressed in a process algebra are composable. In terms of process verification such algebras allow to check control and data flow features of processes based on the algebraically defined operators like livelock and deadlock detection. They also enable to test equivalence of two processes, that is, whether processes behave in the same way in the sense that one process simulates the other and vice-versa. The latter enables organizations to assess whether a partner can replaced by another company that can perform an equivalent job.

A similar suggestion to π-calculus has been made in (Arbab, 2004) in which a language is described for the composition of software components based on the notion of mobile channels. This language, Reo, is a channel-based exogenous coordination model in which complex coordinators, called connectors, are compositionally built out of simpler ones. As such, Reo can be used as a language for coordination of concurrent processes. However, its verification abilities are limited to control flow just like π-calculus. Effort in the same

area is channelled into the Service Mosaic project (ServiceMosaic, 2006). This project aims at developing a platform for modeling, analyzing, and managing web service models including business protocols, orchestration, and adapters. In particular, the focus is on the representation of protocols and their compatibility, equivalence and replaceability (especially in light of the consequences of changes to these protocols for their management). (Nezhad et al., 2006) is representative work produced in this project in which a conceptual framework is presented for analyzing the interoperability of web services. This framework is then utilized to examine existing proposals. However, to a large degree the work done so far seems similar to the use of process algebras to verify and test compatibility and equivalence with regard to control flow features. It remains an open issue as to how verification of other than such control flow requirements can be facilitated.

A very different kind of approach to process verification originates from the field of theorem proving. Theorem proving is the proving of mathematical theorems, typically done by a computer program. For model checking the process is modelled by a set of logical expressions (predicates) specifying its behavior. Then, the properties of this behavior can be verified using the axioms of the particular logic used. One example of such approach is grounded on situation logic. Situation logic (or calculus) is a logic formalism designed for representing and reasoning about dynamical domains. Using second-order logic formulae a dynamic world is modeled as progressing through a series of situations as a result of various actions being performed within the world. A situation represents a history of action occurrences, where each action affects the fluents (or properties) of the world. The world is constrained by formulae about actions (preconditions and effects), formulae about the state of the world, and foundational axioms. Translated to process modeling, a process is defined in terms of second-order predicates. Constraining formulae can then be defined to for example check for deadlock, livelock, and etceteras. Linear logic theorem proving is another kind of solution that has been suggested, e.g. in (Rao et al., 2006) for the composition of semantic web services.

A third strand of work has centered around the idea of constraint satisfaction. Constraint satisfaction is the process of finding a solution to a set of constraints (Apt, 2003). Such constraints enumerate the possible values a set of variables may take. Informally, a finite domain is a finite set of arbitrary elements. A constraint satisfaction problem on such domain contains a set of variables whose values can only be taken from the domain, and a set of constraints, each constraint specifying the allowed values for a group of variables. A solution to this problem is an evaluation of the variables that satisfies all constraints. In other words, a solution is a way for assigning a value to each variable in such a way that all constraints are satisfied by these values. In the context of process modeling constraint satisfaction can be used to verify whether a model meets specified requirements by specifying these requirements as constraints. Alternatively, based on a given set of requirements a solution (i.e. a process model) can be found. An example work following this approach is described in (Aiello et al., 2002) and (Papazoglou et al., 2002) in which users can encode requests as constraints based upon which a service composition plan is generated. For more information on constraint satisfaction in general we refer the reader to (Apt, 2003).

Relevant work can also be found in the area of e-contracts and e-contracting. The

premise underlying such works is that the emerging business to business technologies allow for more automated management of e-contracts including contract drafting, negotiation and monitoring. This has sparked the interest in the formal modeling of contracts as control structures for business collaborations. Such formal modeling then allows organizations to reason about their business collaborations with regard to the contractual agreements as well as to assess whether at runtime everything is done in accordance with the stipulated contract. (Radhakrishna et al., 2005) describes an approach for the conceptual modeling of e-contracts. It also presents a business process model for e-contract enactment in which workflows are generated and executed conform the e-contract. (Marjanovic and Milosevic, 2001) notes that e-contracting must also take deontic obligations and promises into consideration as well as temporal constraint, and the scheduling of activities. Such approach is presented in (Xu, 2004) proposing an approach in which e-contracts are represented using temporal logic. Building on this representation mechanisms are then developed for both the pro-active and reactive detection of contract violations. Although such detection is without doubt useful, it focuses on the monitoring of the execution of contracts after they have been defined. In contrast,we focus on how such contracts can be defined in a consistent manner in particular in light of changing requirements.

All pursued directions have made valuable contributions to the area of business collaboration verification. FSMs, Petri Nets, Actor Models and process algebras have contributed the means to check control flow and data flow properties of processes. More generically, theorem proving and constrain satisfaction based solutions (using e.g. situation or temporal logic) offer the possibility of verifying not only these properties but can also support verification of others like quality of service, payment, security and so on depending on the richness of the models being verified. Thus far however the application of the developed ideas in business collaboration development and management has remained limited. Most works that can be found in literature tend to focus on control flow features whilst ignoring others. Those that do extend beyond that usually limit themselves to for example only workflows (like (van der Aalst, 1998)) or service compositions. As such, what we feel is missing, is the existence of mechanisms for the verification of dependencies among business and technical requirements of processes as well as of dependencies among private and public processes. An exception to the latter is (Dijkman and Dumas, 2004), which defines a mechanism for formal verification of dependencies among service-oriented computing based private processes, exposed protocols and made agreements (i.e. among orchestrations, interface behaviors and choreographies).

2.5 Discussion

In this chapter we have given an overview of a wide range of areas of relevance to dynamic business collaboration development and management. We analyzed the areas of the context, modeling, dynamicity and validation of business collaborations, and discussed the strengths and weaknesses of the works in these domains. Based on these discussions we can now answer the first of our research questions (stipulated in section 1.6), that is, what

is the current state of the art in business collaboration development and management? Grouped by research objective the answer is as follows:

1. **Context Of Business Collaboration**

   The research done thus far on the context of business collaborations, as discussed in section 2.1, is fragmented in nature. Different works place emphasis on different parts of the business collaboration context without acknowledging the existence of other parts and the interrelationships between parts. What is lacking in our view, is a cohesive and comprehensive picture that clearly depicts all the perspectives, aspects and viewpoints of relevance for business collaboration development and management. As a result it is difficult to accurately portray and manage these different perspectives, aspects and viewpoints, the dependencies that exist between them, as well as advanced requirements related to among others quality and security. As such, a first key requirement for our research is to provide such picture. We will address this requirement in Chapter 3 through the introduction of a Business Collaboration Context Framework.

2. **Modeling Of Business Collaboration**

   Extensive work has been conducted with regard to the modeling of business collaborations. The problem in our view with the work reviewed in section 2.2 is the absence of a coordinated approach, which has led to the development of a widely dispersed variety of modeling languages. Each of these languages enables definition of a specific part of the business collaboration context. However, to the best of our knowledge no language (or combination of languages) exist with which collaborations can be modeled in all their details. This is in particularly the case for making dependencies between business and technical requirements, and between private and public processes explicit. To remedy this situation our approach must enable the modeling of the entirety of the business collaboration context. In Chapter 4 we describe how we accomplish this by developing a set of business collaboration models all of which are defined in terms of a generic Business Collaboration Information Model.

3. **Dynamicity In Business Collaboration**

   To tackle the issue of change in business collaboration development and management many relevant suggestions have been made. In section 2.3 we surveyed a number of the submitted proposals from the area of among others workflow and web services. These proposals have typically concentrated on accommodating change within the boundaries of a single (typically private) process. Moreover, usually only one or two types of change are taken into consideration such as only design time changes (i.e. changes in the category of flexibility and/or formal adaptability). As such, although the developed approaches contain interesting ideas none of them in our view adequately and comprehensively address the issue of change for business collaboration. The foremost problem is the limited notion of what changes can occur, most notably

the lack of support for tracing and managing changes among private and public processes, and among business and technical processes. Also, a plethora of mechanisms is currently used to address flexibility, formal adaptability, dynamism and undefined adaptability. We feel that a change handling approach should be uniform in nature, that is, comprise a single solution that handles all types of change in the same manner using the same mechanism. In addition, currently there is little attention paid to the management of changing requirements (rather than the effects of those changes). We will introduce such solution based on rules in Chapter 5 and 6 that addresses these issues.

4. **Validity Of Business Collaboration**

   The ability to formally verify the characteristics of processes has been widely recognized to be of key importance for business process and collaboration automation. Consequently many efforts have been made in this area of research. A diverse range of proposals has been made, roughly falling into the categories of model checking, theorem proving and constraint satisfaction. Important results that have followed include for example the capability to verify control flow like features such as deadlock and reach-ability using an approach like Petri Nets or π-calculus. More generic solutions based on logic or constraint satisfaction can in theory offer more extensive verification options concerning e.g. quality of service, security, payment, and so on. However, how this can exactly be done is of yet unclear. What also emerges from section 2.4 is that the verification of relations between business and technical requirements, and between private and public processes remains largely unaddressed with the exception of works like (Dijkman and Dumas, 2004). We will show in Chapter 5 and 6 how our rule based approach resolves these issues.

Summarizing the above we can conclude that many work has been done from which we can draw in our effort to develop an approach for the dynamic development and management of business collaborations. At the same time though there are a number of gaps that currently exist which stand in the way of the successful realization of such approach. As such, the contribution of the research presented in this dissertation to the existing field(s) will be to: 1) fill the gaps identified in the four points above; and 2) use and merge the resulting new knowledge with existing ideas to construct a solution that enables dynamic business collaboration development and management. In the following chapter, Chapter 3, we will begin with the specification of such solution.

Chapter 3

Context Of Business Collaboration

For me CONTEXT is the key - from that comes the understanding of everything; Kenneth Noland

We are searching for some kind of harmony between two intangibles: a form which we have not yet designed and a context which we cannot properly describe; Christopher Alexander

In the first chapter of this dissertation we provided an introduction to our research, and sketched how we intend to develop a rule based approach for business collaboration that will facilitate their dynamic development and management. Then, in chapter 2 we conducted an extensive literature review to assess the strengths and weaknesses of current solutions. One result of this analysis was the conclusion that cohesive definition of the context of business collaborations has not yet been satisfactorily done. We informally described the business collaboration context already in Chapter 1, where we saw that business collaborations are complex entities. These entities encompass both intra-organizational and inter-organizational processes that interact with each other, where moreover all the processes have a business component as well as a technology component. However, a more structured and unambiguous framework is required to gain a clear understanding of what business collaborations are and in what context they take place. Therefore, the first step towards a rule based business collaboration approach is to analyze the context of business collaborations. This is also illustrated in Fig. 3.1, which shows our current location on the road map described in section 1.3 of Chapter 1 (marked by the square with the bold border).

As can be seen in the figure the result of the first road map step is the *Business Collaboration Context Framework (BCCF).* This framework, as we will see, captures the context of business collaborations in a structured manner by defining a modularized representation of the business collaboration context in order to achieve separation of concern. Coined in (Parnas, 1972) in the context of computer science, separation of concern is defined as 'the process of breaking a program into distinct features that overlap in functionality as little as possible'. A concern is considered to be any piece of interest or focus in a program, usually being synonymous with features or behaviors. The main benefits of separation

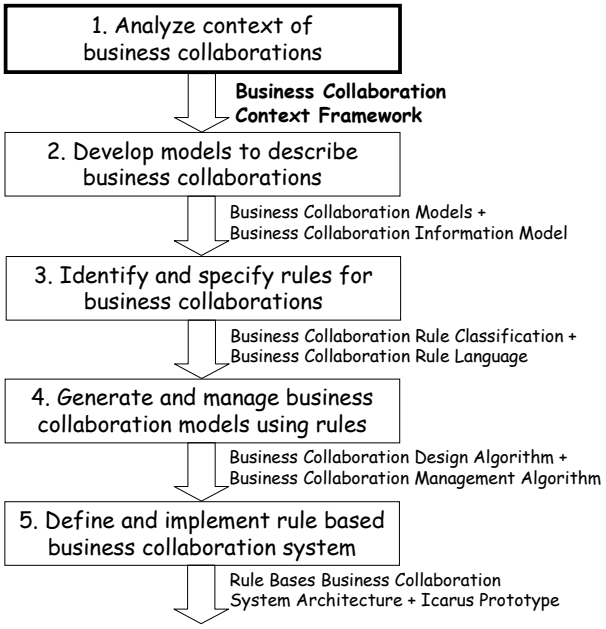

Figure 3.1: Research Road Map - Analyzing Business Collaborations

of concern for business collaboration development and management via modularization are twofold: firstly, and most importantly, modularization helps reduce the complexity of business collaboration development and management by breaking up these collaboration in more manageable chunks. Secondly, it gives us a means with which existing work in business collaboration development and management can be classified, analyzed and compared as has been done in for example (Orriëns et al., 2005).

Concretely, the BCCF modularizes the business collaboration context by adopting a three dimensional view, being *aspect*, *level* and *part*. This modularization is shown in Fig. 3.2.

As can be seen in the figure the BCCF constitutes of three layers separated by a horizontal dotted line, which going from top to down represent *strategic level*, *operational level* and *service level* respectively. Each layer itself is modularized into three aspects, being *internal business process aspect*, *participant public behavior aspect* and *conversation aspect* (denoted by the rounded squared, octagon and hectagon shapes respectively). Finally, each aspect itself is modularized into the five parts of *material part*, *functional part*, *participation part*, *location part* and *temporal part*, which results in every aspect being represented in a '3-D' like manner in Fig. 3.2.

In the following we will further discuss these three dimensions of the BCCF, and explain their purpose and role within the business collaboration context. For illustrative purposes we refer to the AGFIL case study. This case study describes a complex multi-party scenario, which outlines the way in which a car damage claim is handled by an insurance

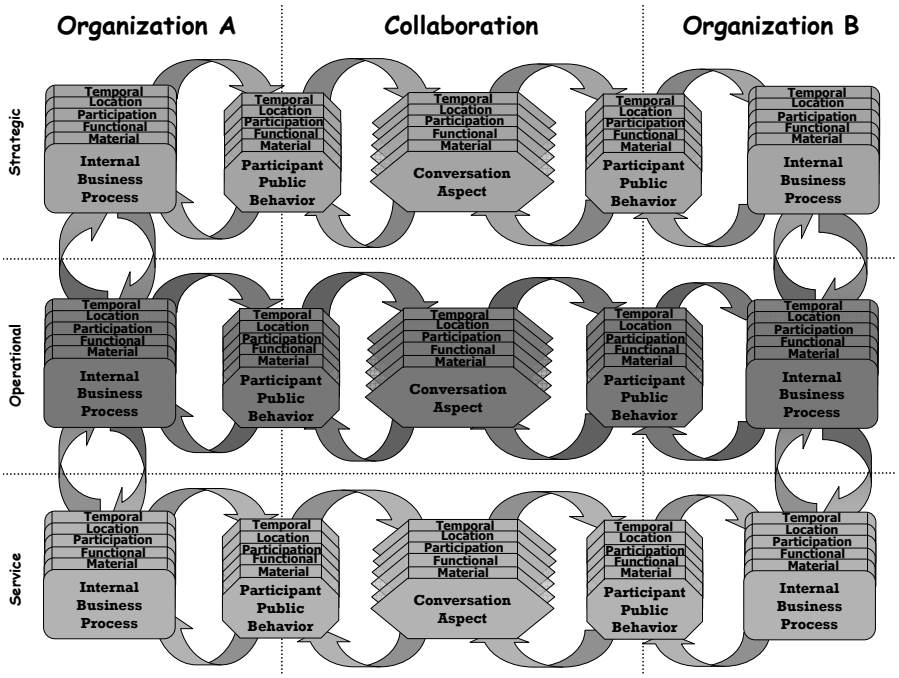

Figure 3.2: Business Collaboration Context Framework (BCCF)

company (AGFIL). AGFIL cooperates with several contract parties to provide a service level that enables efficient claim settlement. The parties involved are `Europ Assist`, `Lee Consulting Services (Lee C.S)`, `Garages` and `Assessors`. `Europ Assist` offers a 24-hour emergency call answering service to policyholders. `Lee C.S` coordinates and manages the operation of the emergency service on a day-to-day level on behalf of `AGFIL`. `Garages` are responsible for car repair. `Assessors` conduct the physical inspections of damaged vehicles and agree repair upon figures with the garages. For a full overview of the case study we refer the reader to Appendix A. The remainder of the chapter is structured as follows: aspects are the topic of section 3.1, levels of section 3.2 and parts of section 3.3. We conclude in section 3.4 with a comparison to existing work. We also provide an analysis of how the BCCF meets the research objective in section 1.4 of Chapter 1 with regard to gaining a clear understanding of the context in which business collaborations take place.

3.1 Aspect

The first dimension, aspect, places emphasis on the different behaviors that an organization exhibits in business collaboration, being the business processes, business protocols and business agreements in which it is involved. Depending on the type of *business collaboration behavior* the purpose and target of development and management varies. The aspect dimension encompasses three types of behavior captured in three corresponding

so-called aspects (inspired by among others (Dijkman and Dumas, 2004), (Peltz, 2003) and (Traverso et al., 2004)): observable, exposed and internal behavior expressed in the conversation aspect, participant public behavior aspect and internal business process aspect respectively. We discuss the purpose and meaning of these aspects in sections 3.1.1 to 3.1.3. Subsequently we explore the relationships that exist between the different aspects in section 3.1.4 (denoted by horizontally curved arrows between the different aspects in Fig. 3.2).

3.1.1 Conversation Aspect

The conversation aspect captures the externally visible behavior between organizations in a business collaboration, i.e the observable behavior. This observable behavior specifies how its organizations are expected to behave in the collaboration from a global, organization independent point of view. The conversation aspect thus defines the behavior agreed upon by the organizations, i.e. constituting a business agreement comparable to those typically found in paper based contracts. The interactions between the different parties in the AGFIL case study, such as `Lee C.S`, `Garage Inc` and `AGFIL` are part of the conversation aspect, describing how these parties have agreed to work together to achieve efficient claim handling.

Agreements are binary in nature, that is, involve two parties. This limits the applicability of the BCCF in the sense that some multi-party scenarios can not be properly captured. However, most of such scenarios can be accommodated through the combination of multiple binary collaborations. To exemplify, the interactions between `Lee C.S` and `Garage Inc`, and between `Lee C.S` and `AGFIL` are in essence conducted separate from one another. Of course they are dependent on one another, however, from the point of view of `AGFIL` it doesn't need to know anything about how `Lee C.S` interacts with `Garage Inc`. All `AGFIL` needs to know is how to interact with `Lee C.S`, where `Lee C.S` is responsible for ensuring that its dealings with `Garage Inc` do not interfere with the agreements made with `AGFIL`. As such, the overall AGFIL multi-party behavior can be expressed in terms of the bi-lateral interactions between the different parties.

3.1.2 Participant Public Behavior Aspect

Rather than being global in nature, the participant public behavior aspect is individual to each organization yet visible to others. Reflecting the exposed behavior of an individual organization, this aspect describes how an organization can publicly behave in a business collaboration, i.e. its potential for cooperating with others as described in its business protocols. As such, it is found at the edge of organizations forming the border with the outside (and therefore placed on the vertical dotted lines in Fig. 3.2 to reflect this). For example, `AGFIL` is capable of interacting with `Europ Assist` to exchange claim information, and with `Lee C.S` to outsource part of the claim management to them. In addition, `AGFIL` will also have the capacity to deal with banks, and many other parties. However, since

these are not of relevance for its realization for efficient claim handling they are part of AGFIL's exposed behavior, but not of the observable behavior.

On the basis of their participant public behavior aspect models organizations can negotiate and define business agreements (i.e. conversation aspect models). AGFIL and Europ Assist e.g. can use their respective representations of their protocols to come to an agreement pertaining as to under which terms they will cooperate to exchange customer claim information. Alternatively, it is possible for organizations to define participant public behavior aspect models to depict how they expect their partners to behave. These expected behavior models can subsequently be published as to inform potential parties of the requirements for cooperation. To exemplify, rather than negotiating AGFIL may simply dictate conditions to Europ Assist by pre-defining the business protocol that the other party is expected to follow.

3.1.3 Internal Business Process Aspect

The internal business process aspect is also individual to each organization. However, it is only of interest to this particular party as it relates to its own internal behavior. That is, the internal business process aspect encompasses the internal activities of participants being its business processes. As such this aspect can not be observed by other organizations, i.e. is private in nature. Garage Inc for example internally will perform a multitude of activities between receipt of a car and reporting repair costs to Lee C.S. These are not of interest though to Lee C.S. And even if they were, then Garage Inc would most likely not be willing to share knowledge about its private processes with Lee C.S (or any other parties for that matter).

As said, the internal business process aspect captures the private activities of organizations. Important to note in this regard is that the scope of this aspect is limited to those private activities that are of relevance to the specific business collaboration under consideration, that is, to those activities that in some manner influence the collaboration and its progress. To illustrate, if we look at the private processes of Lee C.S we can imagine a wide variety of processes such as in the area of salary administration, PR, human resource management, and so on. These however are not directly of interest for Lee C.S collaboration with AGFIL. In that context only the claim management process of Lee C.S is of relevance for this collaboration. All other processes need not be considered. Rather, they will need to be taken into account when Lee C.S wants to capture business collaborations supported by and influencing these processes.

3.1.4 Relations Between Aspects

The conversation, participant public behavior and internal business process aspect are not independent from one another. Business processes are responsible for supporting the behavior the organization is promising to perform in its business protocols. Specifically, a business protocol specifies the conditions under which the inputs and outputs of a business process external to the organization are consumed/produced respectively. As such, every

business process can have multiple corresponding business protocols, where each protocol depicts different conditions to the outside. For example, `Garage Inc` may have several business protocols for handling a damaged car depending on customer status whereas internally the same activities are performed. Vice versa, every business protocol may relate to one or more business processes, e.g. that `Garage Inc`'s car handling protocol relies upon both its private repair and billing process. An important constraint in this regard is that the offered conditions in a business protocol should never be more strict or extensive than those that can be internally supported by the corresponding business process.

Between a business protocol and business agreement a different kind of relationship exists. A business protocol describes how a participant can potentially behave, whereas a business agreement depicts how it is expected to behave. In order for the two to be compatible the manner in which the organization is expected to behave, must be equal to or subset of the way it can behave. In other words, that what is expected of an organization must not exceed its capabilities. To illustrate, if `Lee C.S` expects `Garage Inc` to perform an activity `supply repair information`, this assumes that `Garage Inc` has the capacity to do so. Thus, the activity `supply repair information` must be part of `Garage Inc`'s protocol. A business protocol (or subset thereof) can be part of many business agreements. For example, `Garage Inc` can perform `supply repair information` in other business collaborations than the AGFIL scenario as well. Vice versa, a business agreement corresponds to exactly two business protocols since there are two participants involved in every agreement. For each participant it must be true that the behavior it has agreed to perform, is supported and thus present in its business protocol. In addition, the two protocols must mirror one other in terms of the displayed communication behavior in order to realize actual communication between the two participants. For example, if `Garage Inc` supplies `car repair information` to `Lee C.S`, `Lee C.S` must be able to consume it and vice versa.

It is not difficult to see that changes in either the business process, business protocol or business agreement can have dire consequences. To give a simple example: if, due to new environmental regulations, `Garage Inc` requires more time to assess the damage to a car, then consequently the deadline it promises to `Lee C.S` to give a repair cost estimate is influenced. Such a change in `Garage Inc`'s potential collaboration behavior consequently affects its collaboration agreement with `Lee C.S`. As a consequence a new deadline for car repair estimate provision will need to be negotiated and agreed upon. The reverse is also true: if the agreed upon behavior in conversation aspect changes, then this will prompt an adjustment of the participant public behavior aspects upon which the agreement was defined. Continuing the above example, because of the internal change at `Garage Inc` its agreement with `Lee C.S` requires updating. This in turn forces `Lee C.S` to adjust its participant public behavior accordingly, that is, with regard to its expectations to as to when a car repair estimate will be received. Subsequently this leads `Lee C.S` to a revision of its internal business process aspect to incorporate the new deadline in its claim management process.

Additionally, it is often likely in multi-party business collaborations that changes in

the interactions of two organizations also manifest themselves beyond these interactions affecting the interactions of these organizations with others. This is the case for `Lee C.S` in our example. `Lee C.S` is interacting with several parties within the AGFIL business collaboration, most notably `Garage Inc` and `AGFIL`. These interactions do not take place in isolation. In contrast, they are dependent on one another with regard to their progress. For example, following up after `Lee C.S`'s revision we may find that `Lee C.S` had to redefine its private behavior as well to accommodate the deadline change which originated from `Garage Inc`. These changes to its business processes can then lead `Lee C.S` to revisit the terms under which it offers claim management to others (specifically `AGFIL`). If `Lee C.S` concludes that it can no longer meet these terms and consequently its protocol needs to be adjusted, the change that started at `Garage Inc` is propagated all the way to `AGFIL`. After all, when `Lee C.S` changes its terms for claim management the agreement with `AGFIL` will need to be re-negotiated and re-defined as the original conditions agreed upon are no longer feasible and thus no longer acceptable for `Lee C.S`.

3.2 Level

The second dimension, *level*, recognizes that business collaboration behaviors, i.e. business processes, business protocols, and business agreements, can be observed at several layers of abstraction. The domain, degree of abstraction and the type of developers in development and management varies per abstraction layer. In the BCCF three levels are identified (inspired among others by (Object Modeling Group, 2003a), (Bresciani et al., 2004) and (Zachman, 1987)): the strategic, operational and service level spanning from high level requirements to their support by operational processes to the technical realization of these processes in terms of the used services delivered by the IT-infrastructure. The three levels are the topic of sections 3.2.1 to 3.2.3 respectively. The relations between the different levels are then investigated in section 3.2.4 (denoted by vertical arrows between the same aspects at different levels in Fig. 3.2. Note that for reasons of clarity only the arrows between the internal business process aspect at different levels are shown in the figure).

3.2.1 Strategic Level

At the strategic level the focus is on describing the purpose and high level requirements an organization has with their business collaboration behavior. Depending on the aspect that is considered, developers here are interested in capturing how the organization uses and produces resources to further the private strategies of the organization (internal behavior), the resources it wishes to acquire and/or supply via cooperation with others (exposed behavior), and the resource exchanges to which it has committed itself in actual collaborations (observable behavior). An example is `AGFIL`'s commitment to share the resource `claim management information` with `Lee C.S` in order to help realize its objective of facilitate efficient claim handling.

Thus, at the strategic level the interest is in reasoning about the goals that the organization pursues, and subsequently consider what is the best way to achieve these goals through the exchange of resources. Although such discussion usually takes place in an abstract manner without so much considering actual feasibility, this does not mean that the strategic level will not be detailed in nature. In contrast, high level requirements can also be very specific (albeit usually qualitative) in nature. For example, in their strategic agreement `AGFIL` and `Lee C.S` may address a variety of issues such as pertaining to quality levels, security objectives, payment considerations, and so on, which are to be applicable to their resource exchanges.

3.2.2 Operational Level

The operational conditions under which organizations exhibit their behavior are part of the operational level. This level establishes how organizations conduct their business collaboration behavior on a day-to-day basis. Internally the operational level concerns the routine in which private processes are conducted, such as how and what activities `Lee C.S` performs as it is managing a claim (like contacting the garage, selecting an assessor, and making a final report for `AGFIL`). The behavior exposed to others at this level captures an organization's operational potential to communicate with others, like `Garage Inc`'s capability to provide `Lee C.S` with reports containing car repair estimates. The agreed upon communication between organizations is covered by the conversation aspect at operational level. For example, `AGFIL` and `Europ Assist` will exchange several sources of information, e.g. customer files, claim history overviews, and so on.

As such, developers at this level are not directly interested in the strategic relevance of business collaborations. Rather, they are concerned with how the organization concretely carries out its activities. This is not limited to the functional manner in which things are done e.g. what activities exactly are performed and in which order. It also includes the conditions under which these activities are performed. For example, developers will also deal with what events will be monitored, which data will be used, what quality indicators will be measured, what security mechanisms will be employed, and so on. As a result developers at the operational level of development and management are typically concerned with making the higher level strategic requirements more concrete by putting them into operational terms.

3.2.3 Service Level

The technical requirements of business processes, business protocols and business agreements are addressed at service level. At this level these different business collaboration behaviors describe how organizations use their IT-infrastructure within the context of their business collaborations. We adopt the Service-Oriented Computing paradigm to describe this level (hence the name 'service level') for the reasons as outlined in section 1.1.3 of Chapter 1. In the SOC paradigm an IT-infrastructure is portrayed as providing a range of technical services. For example, for internal usage the IT-infrastructure of `Lee C.S` will

offer such services as claim information retrieval service, car repair estimate processing service, and etceteras in the context of the AGFIL example. Combined these services will form the `claim management service`, which is exposed as a service to other parties (most importantly to `AGFIL`). How this claim management service then interacts with `AGFIL`'s own exposed services is defined in the conversation aspect describing the flow of messages between these different services.

At service level developers will have technical realization as their mind set, that is, how the IT-infrastructure is to be shaped and crafted to enable the organization to do their business in the best possible way. This is a challenging endeavor in particular in light of the stream of constant change to which organizations are exposed. In addition, they focus on technological innovation to give the organization a competitive edge, for example by applying new technology leading to the creation of new business opportunities. Important to note is that the service level is still conceptual in nature, that is, abstracts away from specific technologies, applications and systems like web services, CORBA, J2EE, SAP, and etceteras. This gives developers the freedom to reason about the requirements to be met by the services offered by the IT infrastructure independently of the factual implementation of these services.

3.2.4 Relations Between Levels

Similar to business processes, business protocols and business agreements being related, dependencies exist between the same type of business collaboration behavior at different levels. To start with, the strategic behavior of an organization is typically made more concrete by expressing it in terms of operational activities. One strategic behavior can be mapped to multiple operational behaviors. Vice versa, an operational behavior can support multiple strategic behaviors. For example, `Lee C.S` can operationalize its high level activity of `manage claim` in different ways. `Lee C.S` may for example perform `select assessor` to get an external damage estimate in case a garage's estimate is above a certain threshold. Alternatively, if the estimate is below the set level, then `Lee C.S` may immediately approve by doing `agree repair`. At the same time these operational activities of `Lee C.S` could support other high level steps as well, e.g. `select assessor` is a task that can also be conducted in the context of re-evaluating an earlier claim.

The relations between an operational business process, protocol and agreement and their corresponding service level process, protocol and agreement reflect how operational activities are supported by the IT-infrastructure. Here the relation is many-to-many as well, that is, an operational behavior can be realized in different service behaviors. Vice versa a service behavior can realize multiple operational behaviors. `Garage Inc` may for example use a function `send estimate` provided by its `car repair service` to realize its operational activity `report estimate` to notify `Lee C.S` of the repair estimate height. At the same time though this functionality can also be used to support other activities besides `notify accountant` For example, `send estimate` is also of use to perform the task of notifying the financial department of `Garage Inc` about the made car repair cost estimate in order to allow them to keep track of the budgeted costs for a car repair.

Like the relations between different aspects, the dependencies among levels are also influenced by change. Strategic decisions can affect operational parameters, which in turn can influence demands on the services delivered by the IT-infrastructure. For example, if `Garage Inc` decides to give up its objective of providing car repair estimates in a secure manner, then at operational level the conditions under which activities are performed will change (since there is no longer a need there to employ security mechanisms). Consequently utilization of the IT-infrastructure will become different, as the used services need not exhibit the capacity to enforce security measures. In this way dependencies between levels give organizations the means to assess the feasibility of strategic decisions both from operational and technological point of view. Vice versa, technological changes may prompt adjustment of operational activities, possibly leading to reconsideration of high level strategies. To illustrate, if at technical level `Lee C.S`'s claim information retrieval service becomes unavailable (e.g. because the underlying database crashes), its claim handling activities at operational level will come to an abrupt halt. This in turn will have the affect that `Lee C.S` is no longer able to support its private strategic behavior, i.e. the claim management process.

In addition, when we combine the modularization along level with that along aspect, organizations are empowered to analyze and predict how changes at an individual business collaboration behavior at one level can affect the different types of behavior at other levels. Assuming for a moment that `Lee C.S`'s claim information retrieval service has indeed become unavailable, then it can no longer support its claim management service. As a consequence the flow of information between this service and `AGFIL`'s services will cease to exist, and thus their business collaboration from technical point of view is jeopardized. This means that at operational level both `Lee C.S`' and `AGFIL`'s private activities will need to be paused, which turn will have the consequence that at strategic level the organizations' objectives can temporarily not be achieved. As such, the relations between aspects and levels enable organizations to identify the IT systems that are absolutely vital to their business, and thus whose availability must be ensured at all times.

3.3 Part

The third dimension, *part*, reflects the fact that the different business collaboration behaviors conducted by organizations at the different layers of abstraction cover many different considerations. Parts represent these different considerations by depicting the elements in a business collaboration behavior that have different contexts when observed from different levels. Consequently the focus of business collaboration development and management varies. Five parts are distinguished (inspired by among others (Curtis et al., 1992), (Jonkers et al., 2003), (Nguyen and Vernadat, 1994), (Scheer, 1992), (Vernadat, 1992) and (Zachman, 1987)): material, functional, participation, location and temporal part. Together these five parts encompass all three types of business collaboration behavior at the different levels; where their exact semantics are dependent on the specific aspect and level that they describe. We introduce each part in sections 3.3.1 to 3.3.5 respectively. After that in

section 3.3.6 we review the relationships that exist between the different parts. Note that these relationships are not shown in Fig. 3.2 as to not further clog the picture.

3.3.1 Material Part

The material part emphasizes the structural view of a collaboration behavior, focusing on what 'materials' used to perform a business collaboration behavior; where 'materials' is to be interpreted here in the broadest sense of any tangible and intangible objects This part is akin to the informational view in (Curtis et al., 1992) and (Scheer, 1992), the resource view in (Liles and Presley, 1996), and the data description column in (Zachman, 1987). At strategic level the material part pertains to the resources that are exchanged, like `Europ Assist` capable of providing claim information in its exposed behavior or `Garage Inc` of consuming a damaged car from the customer. Resources can thus be both both physical or informational in nature. At operational level the material part expresses an information oriented view capturing the communication of documents, e.g. that `Garage Inc` sends a report to `Lee C.S` concerning the cost of repairs. A similar view is captured at service level be it in the context of service-oriented computing. Here messages are the exponents of the material part like a `car repair estimate request` to ask for a repair cost estimation.

3.3.2 Functional Part

The functional part concentrates on how a business collaboration behavior is conducted. This part is found in (Zachman, 1987) as the process description column, encompasses the activity and process view in (Liles and Presley, 1996), and is similar to the functional view in (Scheer, 1992). In the BCCF the functional part is concerned with the high level steps that are performed within the collaboration at strategic level, such as the claim managing process of `Lee C.S`. At operational level it deals with the concrete activities performed, like a claim handling employee at `AGFIL` receiving a customer file and a claim history overview as part of the operational agreement. Viewed at service level the functional part pertains to the operations performed by services, for example `get car repair estimate` which is part of `Garage Inc`'s car repair service.

3.3.3 Participation Part

The participation part concerns the participant(s) conducting a business collaboration behavior. In (Scheer, 1992) and (Curtis et al., 1992) this part is not explicitly identified. Rather it is part of their so-called organizational view. However, we feel that these must be separated as they are quite distinct in nature. The organizational view partly deals with geographical location while participation part is focused specifically on the participants involved. Therefore, in the BCCF the participation part represents the stake holders involved at strategic level such as the garage owner of `Garage Inc`. A claim handling employee of `Lee C.S` is an exponent of the part at operational level whereas individual services like `Garage Inc`'s car repair service express the participation part at service level.

3.3.4 Location Part

The location(s) at which business collaborations are carried out, are expressed in the location part. This part is comparable to the network description column in (Zachman, 1987), or the organizational view in (Scheer, 1992) and (Liles and Presley, 1996) (with the exception of the participation details included in the latter view). At strategic level the location part constitutes the organizations that are cooperating like `Garage Inc`. At operational level this part captures the exact units that are involved in the collaboration within these organizations such as `garage repair team`. At service level the location part is specified in terms of the endpoints at which services can be accessed, such as an internet address at which `Lee C.S` can put in a request for a car repair estimate.

3.3.5 Temporal Part

Lastly, the time related dimension of business collaborations is covered in the temporal part. Such part is not really found in works like (Curtis et al., 1992), (Scheer, 1992) and (Liles and Presley, 1996), but it is akin to the time column in (Zachman, 1987). At strategic level the temporal part manifests itself in the schedules that organizations must adhere to while exchanging resources, e.g. that `Europ Assist` must report any customer claims to `AGFIL` within 1 day of its receipt. At operational level it is defined in terms of relevant business occurrences, i.e. business events, such as the receipt of a car repair cost estimate report by `Lee C.S`. Finally, system level events are exponents of the temporal part at service level, reflecting the progress made in the context of the technical services employed. An example is that car repair cost estimate has been entered in the database of `Lee C.S`.

3.3.6 Relations Between Parts

Like aspects and levels, individual parts interact with other parts where each part is related to every other part. The semantics of these interactions are specific to the individual level at which they take place. For example, at operational level the functional part is connected to the participation part expressing who is responsible for carrying out a task. The participation part itself is linked to the location part reflecting the location at which people carry out their work. The location part in turn relates to the material part to indicate how information flows between participants from one location to another. This information is itself used and produced by the tasks that are performed, as such associating the material part with the functional part.

To exemplify, when a claim comes in at `Lee C.S`, and an assessor is required then the one most nearest to the garage of the customer will be contacted. This places restrictions on the relation between the how and who part. Another example is that the time frame to which `Garage Inc` is to adhere concerning provision of the car repair cost estimate, can mean that some extra checks that the garage would normally perform are not done due to the limited time available. In a similar manner material, functional, participation, location and

temporal part are also intertwined at strategic and service level. Together these relations also allow different types of developer to communicate. For example, database specialists and process managers can use the interactions between material and functional part to analyze how their respective domains are related.

In conjunction with the modularization along aspect and level the division into material, functional, participation, location and temporal part becomes even more powerful. Changes in an individual part of a business collaboration behavior can not only be traced in terms of impact to other parts, but also to the corresponding parts in behaviors at the other aspects and levels. For example, suppose that `Lee C.S` makes a change in the order in which its claim management activities are performed (i.e. in the functional part) in its internal business process at operational level. The affect of this change will be (among others) that the temporal conditions (in the when part) applicable to these activities will have to be modified correspondingly. Moreover, though, the influence of the change can be assessed in relation to `Lee C.S`'s private processes at the other levels influencing its and/or the manner in which it makes use of its IT infrastructure. Similarly, the change may lead to adaptations of `Lee C.S`'s potential to collaborate in terms of when the organization carries out which tasks. As a result its agreement with `Garage Inc` would then be affected.

3.4 Discussion

In this chapter we presented a modularized framework called Business Collaboration Context Framework for describing the context in which business collaborations take place. This framework was inspired by many works from literature in a diverse range of research areas as discussed in Chapter 2. The dimension of aspect as it is defined in our framework pervasively throughout the different levels originates from among others the work in (Traverso et al., 2004) where in the modeling of strategic requirements a distinction is made between global versus local requirements, which is comparable to our conversation and internal business process aspect respectively. In (ebXML Initiative, 2002) a differentiation between participant public behavior and conversation aspect is made when covering operational requirements introducing the idea of collaboration protocol profiles and agreements respectively. Finally, at service level (Dijkman and Dumas, 2004) identifies so-called interface behavior, choreography and orchestration viewpoint whereas (Peltz, 2003) discusses orchestrations versus choreography similar to internal business process and conversation aspect. Dependencies among aspects are also present in these works albeit not always in an explicit form.

Also widely established is the idea of utilizing different layers of abstraction. Within business collaboration development and management literature (Bresciani et al., 2004) is a prime example of this encompassing the specification of strategic, business and technical requirements. Another exponent of this school of thought from the area of software development is found in (Object Modeling Group, 2003a) identifying computational independent, platform independent and platform specific layer as of relevance when creating software. Here the first two layers correspond with operational and service level in the

BCCF. These works also recognize the importance of identifying and expressing the dependencies that exist between layers. In fact they consider them to be a crucial factor for success. From the field of enterprize architecture (Zachman, 1987) proposes six layers of abstraction in which owner, designer and builder perspective very roughly relate to our strategic, operational and service level respectively. The reason that we do not adopt any additional levels as is done in (Zachman, 1987), is that even more layering in our view makes the business collaboration context very complex. We feel that as a result the effort required for the development of business collaboration designs to capture this context would spiral out of control.

The concept of part is also crystalized in some form or shape in related research. (Zachman, 1987) is most extensive in this regard with the definition of six so-called facets, being data, process, people, geographical, time and purpose description. These relate by and large one on one to the material, functional, participation, location and temporal parts. Note that the sixth facet in (Zachman, 1987), the why part, is not included in the BCCF as we will utilize the exponents of this part, being rules, in our approach to drive business collaboration development and management. Another work of interest in this regard is ARIS (Scheer, 1992), which identifies an activity, resource, information and system view matching roughly with functional, participation, material and location part. A similar comparison can be made with (Curtis et al., 1992), (Huff et al., 1998), (Jonkers et al., 2003) and (Liles and Presley, 1996) all of which identify similar views as ARIS that are deemed of relevance for perceiving the business of organizations.

As the above demonstrates the BCCF as introduced in this section with its dimensions of aspects, levels and parts is founded on a thoroughly established theoretical basis. As such though at face value it seems as if the BCCF does not contain any new ideas. However, the novelty of the BCCF lies herein that it synergizes the different notions of levels, aspects an parts to establish an overall comprehensive framework for capturing the context in which business collaborations take place. As such, it contributes to the current literature by encompassing and extending existing research. Moreover, the BCCF explicitly relates the ideas that have come out of this research to the field of business collaboration, which is something that to the best of our knowledge has not been done before on such scale and with such scope. The work that comes closest to ours in this regard we feel is the Archimate project (Jonkers et al., 2003). However, in this project the focus is on enterprize architecture whereas we concentrate specifically on business collaboration. Moreover, the parts covered in the BCCF are not all made explicit in (Jonkers et al., 2003).

A shortcoming of the BCCF in line with the set scope restrictions in section 1.5 of Chapter 1 is that it does not explicitly incorporate advanced issues like quality and security. Our vision in this regard is that such issues are orthogonal to levels, aspects and parts, where they have different meaning depending on the exact level, aspect or part. For example, security requirements at strategic level constitute security objectives that organizations have with regard to their business collaborations. In contrast, at operational level such requirements express which security mechanisms will be employed to realize these objectives. Similarly, whereas quality requirements in the business protocols of organizations (i.e. the participant public behavior aspect) constitute promises concerning

quality, in business agreements (i.e. the conversation aspect) they represent agreed upon quality conditions. In relation to the different parts we view advanced requirements as extending these parts. To illustrate, a security requirement in material part at strategic level can convey that a resource must be sent in such manner that it will not be disclosed to others. In this manner the BCCF can easily be extended to include advanced business collaboration requirements.

Summarizing, in relation to the second research question identified in section 1.6 of Chapter 1, being to gain a clear understanding of what business collaborations are as well as reduce the inherent complexity of the context in which they take place, the BCCF contributes in three ways:

1. Firstly, modularization along aspect in conversation, participant public behavior, and internal business process aspect gives organizations the capability to reason about their business agreements, business protocols and business processes. Moreover, it allows them to consider dependencies among these processes, protocols and agreements. This in turn empowers organizations to reason about relations between their individual business collaborations with parties by tracing dependencies from one business agreement to another via the intermediate business protocols and business processes. As organizations have constructed extensive networks with other organizations (like supply chains or value webs) to deliver their corporate business services, such reasoning capability is crucial in order for them to successfully manage their business collaborations.

2. Secondly, modularization along level in strategic, operational and service level assists organizations to distinguish between the different layers of abstraction at which their business collaboration behaviors take place. Moreover, by recognizing the dependencies between these layers organizations can reason about them, and communication among different types of designers can be established. This enables organizations to align their business and IT. This is crucial as organizations nowadays rely virtually completely on their IT-infrastructure to operate their business. In addition, in combination with the modularization along aspect organizations are able to reason about how different types of behavior at different layers of abstraction influence one another; which is of vital importance as cooperations between organizations are both business and technical in nature.

3. Thirdly, modularization of the context in which business collaboration behaviors along part into material, functional, participation, location and temporal part gives organizations the opportunity to cover each of the different parts of these behaviors individually. At the same time organizations can take the relations of one part with the other parts into consideration. This allows different types of developer to be involved in the design process such as data specialists, process managers, and etceteras. Communication between them can be facilitated by utilizing the part interactions. As such, modularization along part helps organizations deal with the inherent complexities of their behavior in the different aspects at each of the distinguished levels.

In short, the developed BCCF provides us with a clear understanding of the context in which business collaborations take place by usage of three dimensions; as such reducing the complexity of these complex entities. The next step is to develop the means with which this context can be captured. The approach that we developed for this purpose is model based in nature, and is the topic of Chapter 4.

Chapter 4

Modeling Of Business Collaboration

All models are wrong, but some are useful; George Box

When I model I'm pretty blank. You can't think too much or it doesn't work; Paulina Porizkova

The BCCF framework presented in Chapter 3 provides a cohesive description of the context in which business collaborations take place. Its modularization helps reduce the level of complexity of business collaborations by partitioning this context in the dimensions of aspects, levels and parts. As such, we have completed the first step in the road map outlined in section 1.3 of Chapter 1. The next step, as illustrated by Fig. 4.1, is to develop the means with which organizations can make the context of their business collaborations explicit. The result of this step will be a model based approach comprising a set of business collaboration models and an abstract *Business Collaboration Information Model*. As we will see in this chapter these in combination enable organizations to specify their business collaborations in an explicit and uniform manner.

Concretely, based on the BCCF framework the model based approach is about capturing the parts at different levels and different aspects for each individual business collaboration. As a result, several *models* and *mappings* (expressing dependencies between models) can be generated based on the level and the aspect they represent. Fig. 4.2 provides an overview of the different models and the relations between them. Note that for reasons of clarity the modularization into parts of the different models has been omitted from the figure. As can be seen the figure is divided in three sections representing the private, exposed and observable behaviors of organization A and B respectively (in line with the aspect dimension in the BCCF). The rounded rectangles represent the internal behaviors of party A and B at strategic, operational and service level respectively (following the level dimension in the BCCF), and are referred to as *processes*. The octagons capture the exposed behaviors of A and B referred to as *protocols*, whereas the hexagons express the observable behavior between both parties referred to as *agreements*. Vertically directed curved arrows represent *vertical mappings* expressing dependencies between strategic, operational and service processes, protocols and agreements, whereas horizontally directed curved arrows

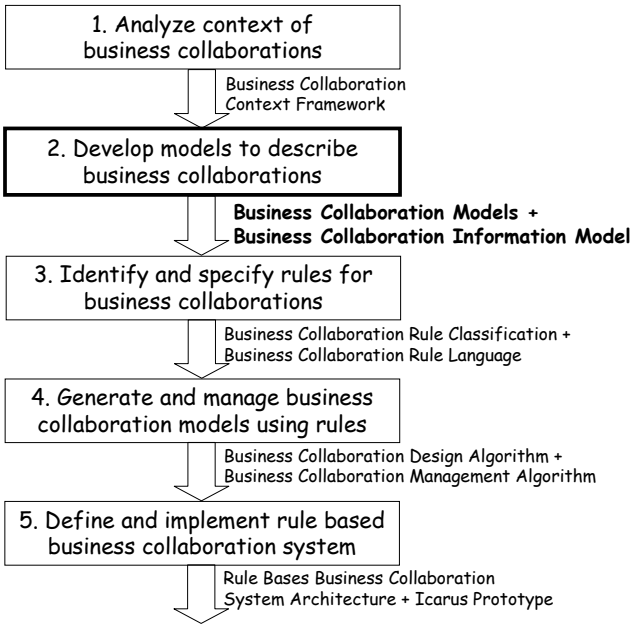

Figure 4.1: Research Road Map - Modeling Business Collaborations

express *horizontal mappings* to capture relations between private processes, protocols and agreements.

Models consist of modeling elements which express the different parts. By enriching modeling element definitions with additional characteristics advanced requirements such as security and quality can be expressed (in line with the remarks in this regard in section 3.4 of Chapter 3). Models are loosely based on UML conventions, that is, in order to distinguish between elements expressing different parts, we represent them in different shapes in their models: material part is shown as folded corners, functional part as rounded rectangles, participation part as octagons, location part as plaques, and temporal part as hectagons (see also the legend at the bottom of Fig. 4.3). The mappings between models relate these different elements (and their properties) to capture the dependencies among levels and aspects identified in section 3.1.4 and 3.2.4 of Chapter 3 respectively. Relations among different elements within the same model convey the dependencies between the different parts as discussed in section 3.3.6.

In this chapter we will explain the purpose and workings of the different models for business collaboration representation. The chapter is structured as follows: we first introduce the modeling constructs with which the strategic, operational and service level requirements of business collaborations can be captured in sections 4.1 through 4.3 respectively. We will also briefly show how these constructs can be enriched to express advanced requirements such as quality and security characteristics. Subsequently, we will make clear how the dependencies between these requirements at different aspect and level

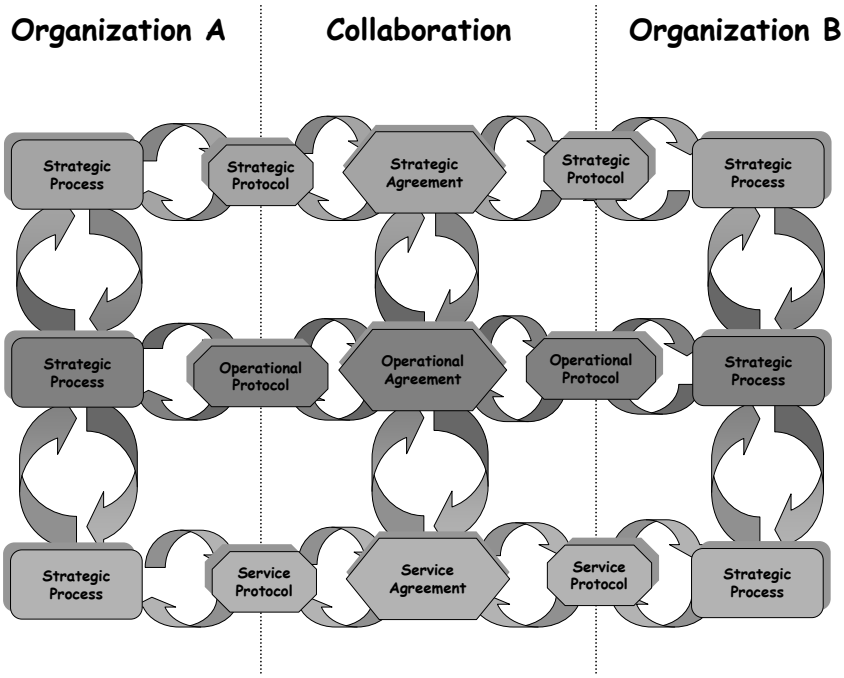

Figure 4.2: Modeling The BCCF

can be made explicit in section 4.4 and 4.5 respectively. After that we present the Business Collaboration Information Model (BCIM) which provides the meta-model specification of the different individual models, and mappings between them. Finally, we conclude by providing a summary of the model based approach and an assessment of its merits in relation to the research objectives we identified in section 1.4. Snippets of exemplary models for the AGFIL case study are provided in Fig. 4.3. These example models describe a part of the cooperation between `Garage Inc` and `Lee C.S` in which `Garage Inc` notifies `Lee C.S` of the estimated cost for a car repair. The models display the strategic, operational and service representation of this interaction at the different aspects. More extensive models for the AGFIL case study are provided in Appendix A.

4.1 Strategic Models

At strategic level, *strategic models* capture purpose and high level requirements of business collaborations, akin to requirements analysis done in e.g. (Bresciani et al., 2004) and (Traverso et al., 2004). Enterprizes can define these models to capture the requirements of their business collaborations, i.e. to make the strategic details of their business collaborations explicit. This enables organizations to manage their cooperations with others in terms of strategic relevance and purpose. As exemplified in Fig.4.3 strategic models such as `AGFIL-STM` are expressed in terms of resources, steps, stake holders, organizations, and

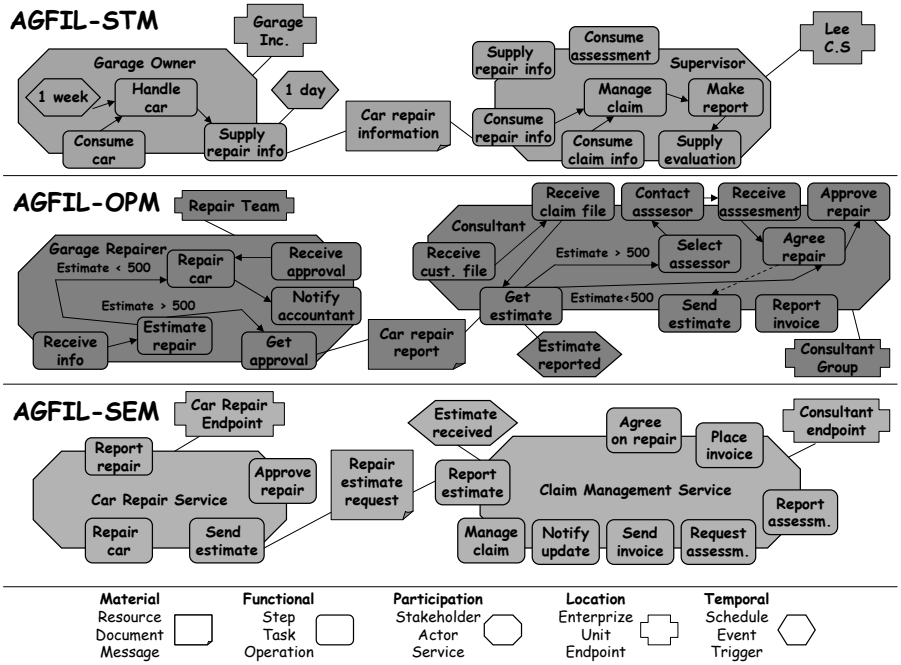

Figure 4.3: AGFIL Example Models

schedules to capture the basic business collaboration requirements at strategic level. An overview of these building blocks and their relations are shown in Fig. 4.4 in an UML class diagram like style.

As the figures show **resources** make up the material part of a strategic collaboration, where each resource (such as `car repair information`) provides an abstraction mechanism for means such as financial, human and informational capital. Resources have a 'name', 'value', 'capacity' and 'description'. Resources are 'drawn on by' and 'produced by' **steps** which represent high level functions such as `manage claim`. Observe that these as well as all other discussed relations can be from two directions. For example, from the point of view of steps a step 'draws on' and 'produces' a resource. This is conveyed in the different figures by labeling each end of an association with different role names.

Continuing the discussion of Fig. 4.4, steps are part of the functional part, and can be 'dependent on' one another. Parallel steps are implicitly specified by having two steps be dependent on the same preceding step, but not on each other. If a step is not dependent on another step, it is regarded to be the initial step in a business collaboration behavior. If a step does not have another step that is dependent on it, then this step is considered to be a final step in a business collaboration behavior. Steps are of type 'internal' (like `handle car` in Fig.4.3)), i.e. private to an organization, or of type 'supply'/'consume' representing resource supply and consumption respectively (e.g. `consume repair information`). Observe that strategic processes contain a mix of private and communication steps, where the latter represent the points at which interaction with the outside takes place. In contrast,

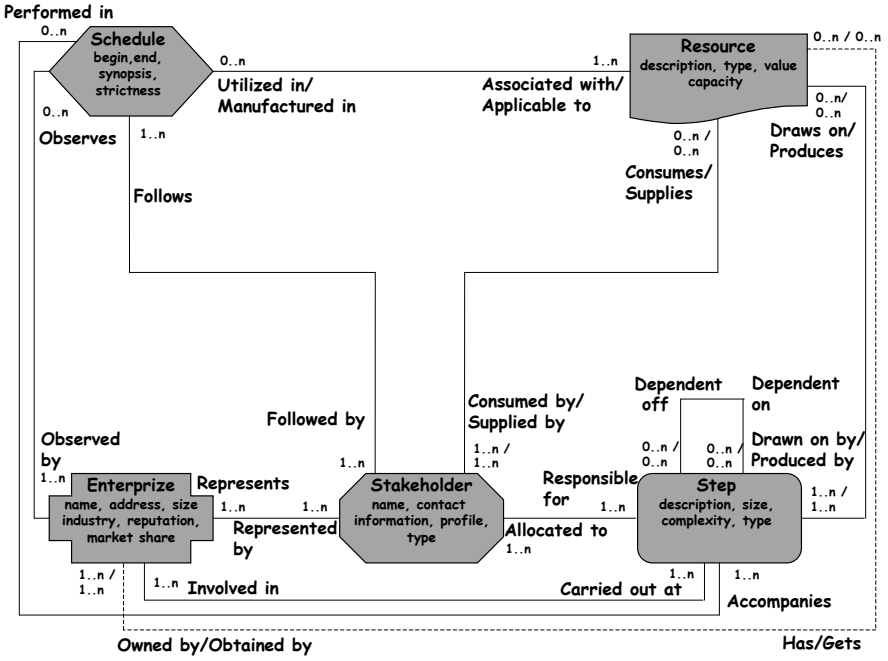

Figure 4.4: Strategic Model Building Blocks

strategic protocols and agreements only constitute of communication steps.

Stake holders like `garage owner` describe the participants who are responsible for 'carrying out' defined steps, i.e. encompass the participation part. Stake holders are characterized by their 'name', 'function', 'contact information', 'background', and so on. Stake holders 'belong to' **organizations** (like `Garage Inc`) identifying the organizations involved. As such, organizations express the where part detailing their 'name', 'address', 'contact information', 'size', 'reputation', 'industry', 'mission statement', etc. Stake holders and their organizations 'adhere to' **schedules** reflecting temporal constraints like the deadline of `1 week` for `handle car`. Schedules express the when part. They have a begin and end date, and a level of strictness. This level indicates how strict the schedule is to be followed. Some schedules may be rather loose whereas others are perhaps the result of legislation, and thus must be strictly adhered to by the organizations involved.

Using the modeling elements of resources, steps, stake holders, organizations and schedules, organizations can define their different strategic collaboration behaviors. Enterprizes can model the conversation aspect to define the exchange of resources between organizations to achieve shared strategic objectives, i.e. a strategic agreement similar to for example i$*$ models in (Yu, 1997). Fig. 4.3 represents such an agreement between `garage owner` and `supervisor` to exchange `car repair information`. To define its strategic collaboration potential, i.e. its strategic protocol, an organization specifies its capabilities in terms of the resources it can exchange. The organization can capture its internal business process aspect by identifying the high-level steps performed to realize this potential. To illustrate,

the exposed behavior of `garage owner` in Fig. 4.3 depicts that it can exchange `car repair information` to which end `garage owner` internally carries out `handle car`.

The strategic model constructs presented thus far can be enriched with additional properties to express more advanced requirements. For example, a quality characteristic may be added to **steps** to express the degree in which a request for carrying out particular steps actually results in their performance. For example, `garage owner` may require that the step `consume repair information` of `Lee C.S` is fully accessible (i.e. it can be performed) between 09.00 to 17.00 on a weekday. Specification of security requirements may be facilitated in a similar manner. To exemplify, **resources** can be enriched with an 'unauthorized disclosure' property, which if set to 'false' means that the exchanged resources must only be disclosed to authorized parties. The `car repair` information send by `AGFIL` to `Lee C.S` is an illustrative example of such resource within the AGFIL example. A full fleshed discussion of the specification of advanced requirements is beyond the scope of this dissertation. (Orriëns, 2006a), (Orriëns, 2006b), (Orriëns, 2006c) and (Orriëns, 2006d) offer starting points for readers interested in this matter.

4.2 Operational Models

At operational level, *operational models* like the `AGFIL-OPM` in Fig. 4.3 depict how activities are carried out in a business collaboration. Operational models help organizations to capture these operational activities, i.e. make their daily business routines explicit. This in turn enables them to visualize, analyze and if necessary change these routines. The basic requirements of business collaboration are expressed at operational level in terms of documents, tasks, actors, units, and events. These express material, functional, participation, location and temporal part at operational level respectively. Their relations and basic properties are shown in Fig. 4.5.

Documents like `car repair report` in Fig. 4.3 represent the flow of information in a collaboration behavior, and capture the material part in operational semantics. Documents constitute abstract information containers, and provide characteristics of the information e.g. in terms of 'language', 'semantics' and 'syntax' used. Documents are internally comprised of parts that represent snippets of data. Documents are 'used by' and 'produced by' **tasks**, which represent specific business functions (alternatively interpreted as that tasks 'use' and 'produce' documents). Tasks model the functional part, and are of type 'internal' or 'communication'. Internal tasks constitute private activities, e.g. `collect claim form` done by `claim office employee` whereas communication tasks involve receipt or sending of information like `receive customer file` and `provide estimate` (as indicated by the task type set to either 'receive' or 'send' respectively).

Actors such as `garage repairer` and `consultant` are 'responsible for' carrying out tasks. Actors represent the participation part, concretely the human beings involved in, exhibiting characteristics like name, function, phone number, email address, and home page. Actors 'belong to' units, which are part of the location part. For example `garage repairer` is part of unit `repair team` unit. Units themselves have details such as name,

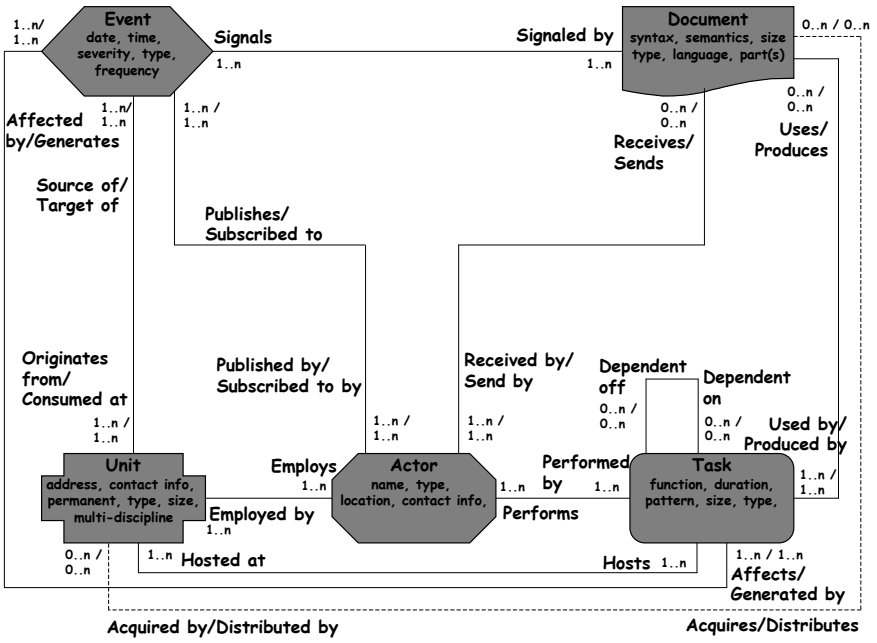

Figure 4.5: Operational Model Building Blocks

address, address, home page, fax, telephone, email, and etceteras. The temporal part is covered by **events**, which are used to describe business occurrences. Events have properties such as date, time, severity, and frequency, and are used to assess progress, keep logs to ensure non-repudiation, and so on. Events are 'signalled by' documents, and as such 'originate from' and are 'generated by' tasks.

In terms of the aspects an operational model constitutes the following: in the conversation aspect operational models constitute agreements which define the observable flow of information between actors comparable to RosettaNet (RosettaNet, 2006) or ebXML BPSS (ebXML Initiative, 2006) models. Fig. 4.3 thus represents the operational agreement made between `Garage Inc` and `Lee C.S` depicting how and under what conditions communication between the two organizations is to take place. In the participant public behavior aspect an operational model constitutes an operational protocol, which defines the documents an actor can exchange like `garage repairer` capable of sending `car repair report`. Such models are similar for example to ebXML CPP based models (ebXML Initiative, 2002). In the internal business process aspect the resulting models specify workflow like business process descriptions (akin to e.g. BPML (Business Process Modeling Initiative, 2002)), for example describing the private activities of `consultant`.

Like strategic model constructs, operational model constructs can be extended to reflect advanced properties. For example, to facilitate the assessment of quality objectives indicators can be defined for the different classes. To demonstrate, for **tasks** class we can add a property 'accessibility rate', which indicates the rate at which tasks can be accessed.

To illustrate, `Lee C.S` may set the rate for `get estimate` to 90%, i.e. that nine out of ten requests are honored. Similarly, efficiency of a step can be expressed in terms of task 'throughput', reflecting how many times a task can be performed in a specific time period. Concerning security, at the operational level defenses against the identified threats at strategic level are established. To exemplify, a threat like masquerading can be prevented through the usage of authentication. To illustrate, whenever a `garage repairer` at `Garage Inc` sends a car repair estimate to an accountant at `Lee C.S`, he/she has to authenticate himself/herself. Such dependencies among advanced requirements at different levels can be expressed and enforced using so-called control rules (discussed in section 5.3.1 of Chapter 5). For more information on advanced requirements specification in operational models the reader is referred to (Orriëns, 2006a), (Orriëns, 2006b), (Orriëns, 2006c) and (Orriëns, 2006d).

4.3 Service Models

At service level, an organization develops *service models* to describe its IT-infrastructure in terms of the services that the systems and applications in this infrastructure offer. This allows the organization to analyze and describe its technical capabilities, capture dependencies among systems and applications, identify which systems and applications are crucial, and so on. Note that, as observed in section 3.2.3 already, service models are specified independent from any specific technology. However, their make-up is such that they can be easily transformed into web services based specifications to make them executable. Having said that, service models are defined in terms of messages, operations, services, endpoints, and triggers to capture the basic requirements of business collaborations at service level, as exemplified by the `AGFIL-SEM` in Fig. 4.3. Fig. 4.6 shows these building blocks and their interdependencies.

In Fig. 4.3 and Fig. 4.6 messages represents the material part, and represent containers of information (e.g. `repair estimate request`), consisting of meta-data and actual data contained in their headers and body respectively. Meta-data comprises the information required to deliver the message and enable its processing (like parameters concerning reliable messaging, encryption styles, characters used, etc). Payloads contain any content of the message not conveyed in its meta-data (like text documents, images, video files, etc). Messages function as the 'inputs' and 'outputs' of operations such as `report estimate` (i.e. the 'input of' and 'output of' operations). Operations represent specific technical functions as such capturing the functional part. Operations are described in terms of their access details (like 'access point). Operations, just as steps and tasks at strategic and operational level respectively, can be dependent on one another. Additionally, they are either of type 'internal' or they constitute communication activities, where the options are 'input', 'output', 'input-output' or 'output-input' reflecting the four typical operation interaction types (as identified also in WSDL (Christensen et al., 2001)).

Operations are 'grouped in' services (e.g. `car repair service`), which are collections of logically related operations. Services capture the participation part at technical level,

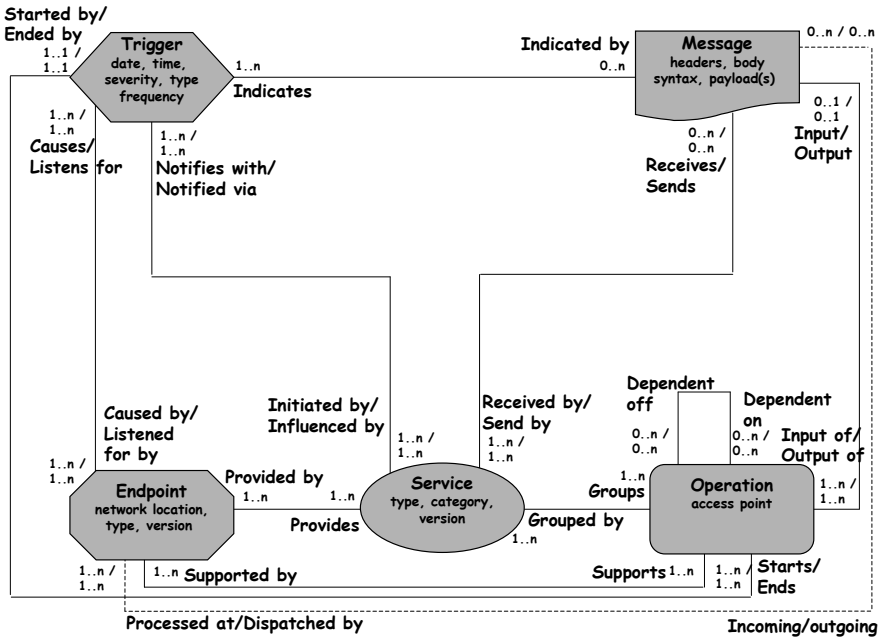

Figure 4.6: Service Model Building Blocks

and have attributes like name, category, version, where category indicates what type of functionality the service provides (for example in context of the North American Industry Classification Scheme (NAICS)). Services themselves are 'provided by' endpoints like `claim handling endpoint`. Endpoints encompass the location part and have properties network location and type. To express technical occurrences triggers like `claim request acknowledged` are defined. Triggers constitute the temporal part in the context of the service-oriented computing paradigm. They are somewhat similar to faults in WSDL (Christensen et al., 2001), though more extensive in scope as they encompass both normal as well as exceptional occurrences. The difference between the two is reflected in the severity of the trigger. Other trigger characteristics are date, time and frequency.

By combining messages, operations, services, endpoints and triggers organizations can define their observable, exposed and internal behavior at service level. Underlying observable behavior service models is the notion of choreography (Peltz, 2003) defining the agreed upon exchange of messages among services like between `car repair service` and `claim management service`. These models are similar to those that can be developed with for example (Banerji et al., 2004) and (Little et al., 2005), and describe the message interactions among a set of services. Models of the participant public behavior depict service protocols, defining the operations a service can offer and the conditions under which this can be done (akin to e.g. a WSDL service description (Christensen et al., 2001) or a WSCI service definition (Arkin et al., 2002)), like `car repair service` offering `send estimate`. Within a service the modeling elements depict internal behavior models akin to

orchestration (Peltz, 2003) (not shown in Fig. 4.3). The most notable specification which partially resembles this type of model is BPEL4WS (Curbera et al., 2002).

In service models advanced requirement specification can be accommodated by extending the service model constructs with appropriate properties. For example, the accessibility rate of a task at operational level may be calculated at service level for each operation by dividing the 'number of accepted requests' (i.e. handled messages) by the 'total number of requests' made over a given period of time. To illustrate, given that `get estimate`'s rate is set to 90%, operation `report estimate`'s ratio of accepted requests must be above that. Security related properties can also be included to define how defense mechanisms proposed at operational level will be implemented. To illustrate, to implement the required authentication for `garage repairer` the `car repair service` may depict that some proof of knowledge must be provided as input like a username/password combination, date of birth, or pin code.

A concluding remark: in line with the observation in section 3.2.3 of Chapter 3 service models are conceptual in nature and thus do not represent executable definitions. This is conform the scope we defined in section 1.5 of Chapter 1, where we excluded the actual implementation of business collaborations. However, as this is ultimately required let us shortly put forward our vision on this: in general service models will have to be mapped to executable specifications depending on the wishes of individual organizations with regard to the preferred implementation technology. An obvious candidate for such technology though is web services based technology. Service models can be implemented using such technology by mapping the services in these models to web services described by WSDL (Christensen et al., 2001). The mappings will entail the mapping of operations to WSDL operations, messages to WSDL messages, endpoints to WSDL ports and services, and triggers to WSDL faults. Based on these mappings invocation of operations can then take place by interpreting the service models and perform invocations through the mapped web services as prescribed by the models. In this relatively simple manner service models can be transformed into executable representations.

4.4 Vertical Mappings Between Models

As we observed in section 3.2 of Chapter 3 the business processes, business protocols and business agreements of organizations at different levels are related to each other. For the specification of such dependencies between these different business collaboration behaviors at different levels we employ vertical mappings. Vertical mappings, as described in (Orriëns and Yang, 2005), facilitate alignment between strategic and operational, and operational and service behaviors thus bridging the gap between high level business requirements and their technical realization. Vertical mappings are realized by providing links between the classes and their properties in the different models at the different levels. The vertical mappings among these classes and properties are based on the implicit links that exist between classes that describe the same part at different levels in the same business collaboration behavior. With such mappings in place organizations are able to make

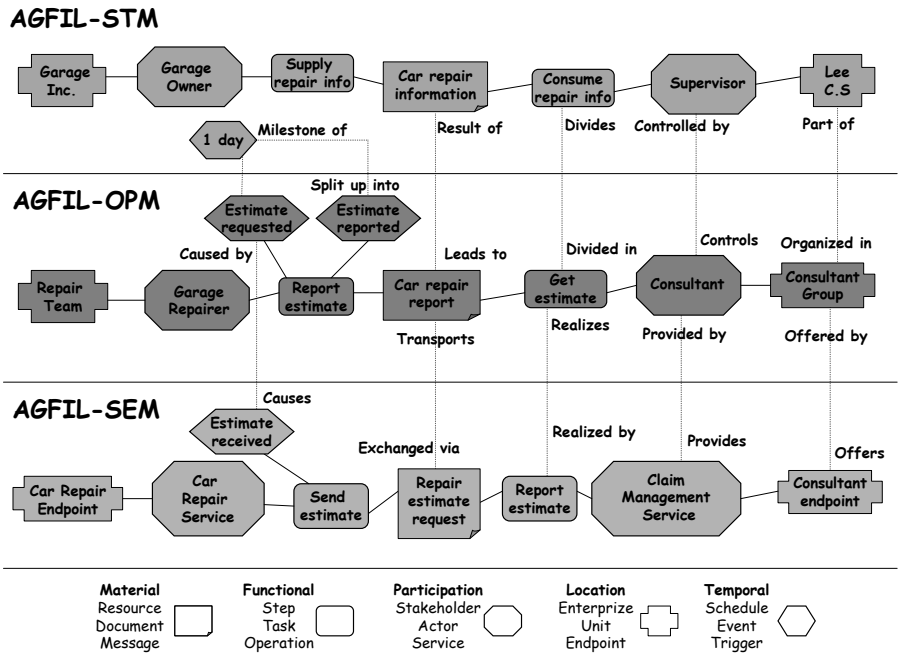

Figure 4.7: AGFIL Vertical Mappings

dependencies among their business processes, protocols and agreements at different levels explicit. This in turn allows them to reason and communicate about these dependencies, trace the affect of changes from one level to another, and as such facilitate the management of these changes.

Concretely, we define five vertical mappings, which are the topic of section 4.4.1 through 4.4.5 respectively. In these sections we also discuss several dependencies among the properties of classes to demonstrate how these influence the consistency of the vertical mappings. The reader is to be aware though that this latter discussion is intended to be illustrative in nature rather than exhaustive. Throughout the following we will refer to the example vertical mappings provided in Fig. 4.7 for the agreed upon behavior between `Garage Inc` and `Lee C.S`, where mappings are denoted by dotted lines. Note that each example mapping has two labels indicating the role played by each class in this relation (identical to the usage of links in individual models).

4.4.1 Material Part

Resources at strategic level are mapped to documents at operational level via a 'leads to' relation. Each resource is mapped to at least one document, whereas each document can be the 'result of' to multiple resources. The idea is that when a resource is exchanged this is accompanied by an information flow. It is this information flow that is defined in terms of document(s) at operational level. An example is the exchange of `car repair information`

which leads to the communication of `car repair report` in Fig.4.7. Other documents that might be exchanged can be photos of the car, replacement part descriptions, and so on.

Documents themselves are mapped to messages using 'exchanged via' relations, which reflect how the information flow is supported in the IT-infrastructure. The communication of `car repair report` is requested for by sending `car repair request` for example in Fig.4.7. The report is then send in response via `car repair response`. A document is exchanged via at least one message, where one message can 'transport' multiple documents. `car repair report` is transported e.g. with `car repair response`, but may also be the payload of other messages send. Vice versa, each message can facilitate exchange of multiple documents as its payloads. For example, `car repair response` can have as payloads the `car repair report`, but also photos and video of the damage, and so on.

When mapping resources to documents and then to messages, the individual properties of these elements must be taken into account. To demonstrate, suppose that `Garage Inc` must send `car repair information` to `Lee C.S` in such manner that it is not disclosed to others. Then, at operational level `Garage Inc` must stipulate that `car repair report` is send in a confidential manner by applying e.g. cryptographic and/or hashing techniques to this document. As a result, at service level this means that `repair estimate request` must use some form of stream protocol or cypher protocol, depending on whether a cryptographic or hashing technique is preferred at operational level. If this is not the case, then mappings between these elements are not feasible.

4.4.2 Functional Part

Steps represent high level strategic activities. By mapping them to tasks these steps are operationalized. To illustrate, `supply repair information` is mapped via a 'divided in' relation to `report estimate`. Each step is decomposed into at least one task, but usually more as steps are likely to be very abstract in nature. For example, the exchange of car repair information may require `repair team` to send multiple documents like the actual report, photos of the damage, and so on. A task can 'divide' multiple steps, reflecting the idea is that specific business functions are of use in the context of multiple high level activities.

Tasks are themselves mapped to operations using 'realized by' relationships. These relationships indicate how these tasks are supported via the operations of technical services provided by the IT-infrastructure. A task is realized by one or more operations. Vice versa an operation can help realize multiple tasks. For example, task `report estimate` is facilitated technically by operation `send estimate` of service `car repair service`. At the same time `send estimate` may also be used to realize other tasks like `repair team` notifying the financial department of the estimated repair cost.

Both types of mapping must also take the properties of the involved elements into consideration. To demonstrate, suppose that `garage owner` requires that `consume repair information` is fully accessible (i.e. it can be performed) between 09.00 to 17.00 on a weekday. `Lee C.S` has to set the rate for `get estimate` to 100% for those conditions. Then, given that `get estimate`'s rate is set to 100%, operation `report estimate`'s ratio

of accepted requests must be equal to that. Otherwise, there is a conflict between the quality objectives that an organization has on strategic level with its quality indicators at operational level and the measurements performed at service level. As such, no mappings between the involved elements can be defined.

4.4.3 Participation Part

The stake holders at strategic level are mapped to actors at operational level via 'controls' links. The purpose is to make explicit how a stake holder delegates the realization of the high level steps it is responsible for to the actors it controls (such as `garage owner` delegating car receipt and repair to `garage repairer`). A stake holder will typically control multiple actors with a minimum of one actor. In addition to `garage repairer` the `garage owner` will also employ other personnel like accountants, salesmen, and so on. Each actor may be controlled by multiple stake holders. Typically though an actor will be controlled by one stake holder, since otherwise confusion may arise when an actor receives conflicting instructions from different stake holders.

Actors themselves are portrayed as services at service level, like `garage repairer` connected to `car repair service` via a 'provides' mapping. Each actor provides one or more services, where each service is provided by exactly one actor. Note that the mapping of actors to services encompasses both human and non-human based services. For example, internally `car repair service` may be performed by `garage repairer`, but due to the self-contained nature of services from the point of view from the IT-infrastructure it is just another service being offered; where `garage repairer` e.g. sends car repair costs to the `car repair service`. Alternatively, `car repair service` might be supported by an application as such automating this part of the actor's activities. This transparency of services allows organizations to develop and manage both automated and non-automated behavior in the same manner.

4.4.4 Location Part

Enterprizes are organized in units at operational level using 'organized in' links, like `Garage Inc` having a unit `repair team`. The purpose of this mapping is to mimic the manner in which organizations usually set up their organizational structure. In accordance with the latter an organization constitutes one or more units. Each unit may optionally be subdivided into smaller units, for example `repair team` consisting of a welding group and tyre replacement team. An unit itself belongs to exactly one organization akin to the structure of organizations in real life.

At the service level, units such as `repair team` are mapped to endpoints like `car repair endpoint`, expressing that an unit is responsible for providing an endpoint in the IT-infrastructure. To capture such a relationship 'offers' mappings are utilized. Each unit can offer many endpoints, but each endpoint is offered by exactly one unit. Note that this does not mean that `repair team` will have to maintain `car repair endpoint`. This might well be done by the IT department. However, `repair team` is responsible for delivering

the functionalities provided by this endpoint; in the example to provide estimates on car repair costs, that is, to offer `car repair service`.

4.4.5 Temporal Part

Schedules expressing temporal requirements at strategic level are 'split into' events representing concrete business occurrences at operational level, e.g. `estimate requested` indicating a car repair estimate request has been made. Events thus make a schedule more concrete by dividing it into business occurrences, which allows progress to be monitored. Each schedule is split into a single or multiple events while an event can be part of multiple schedules. In Fig. 4.7 schedule `1 day` is split into two events with `estimate reported` in addition to `estimate requested`. Note that the characteristics of the schedule and its events must be in sync. To illustrate, given the period of 1 day the time elapsed between the occurrence of `estimate request` and `estimate reported` must not exceed 24 hours.

Events are themselves mapped to triggers like `car estimate reported` 'causes' `car estimate received`. These mappings express how occurrences at service level (i.e. in the IT infrastructure) relate to operational level events. Usually an event is mapped to many triggers, i.e. before a business event occurs multiple technical triggers must have taken place (although in some cases a single trigger may suffice). A trigger itself can be mapped by multiple events, that is, an occurrence at service level can be of relevance for more than one business event. Similar to the relation between a schedule and its events, the date and time at which triggers occur must be consistent with the event to which they belong. For example, if `car estimate reported` is to occur at the latest on February 1 at 07.00 pm, then this means that trigger `car estimate received` must be observed before or at this moment in time.

4.5 Horizontal Mappings Between Models

In the previous section we discussed making explicit the dependencies among the same business collaboration behavior at different levels. In section 3.1.4 we observed that there also exist between different business collaboration behaviors of organizations at the same level. In order to enable organizations to manage these interrelationships, we utilize horizontal mappings. Horizontal mappings allow organizations to maintain compatibility between their internal business processes, exposed protocols and business collaboration agreements. Horizontal mappings define links between the same types of modeling element, which are part of models describing these different business collaboration behaviors. The mappings are grounded on the discussed relations that exist among private processes, exposed protocols and made agreements at a particular level. The purpose of the mappings is twofold: 1) to capture how the communication activities in a business process relate to those in the corresponding business protocol; and 2) to describe how a business protocol supports the agreed upon communication activities of the organization in a business agreement. By making these dependencies explicit organizations can reason about them. Moreover, the

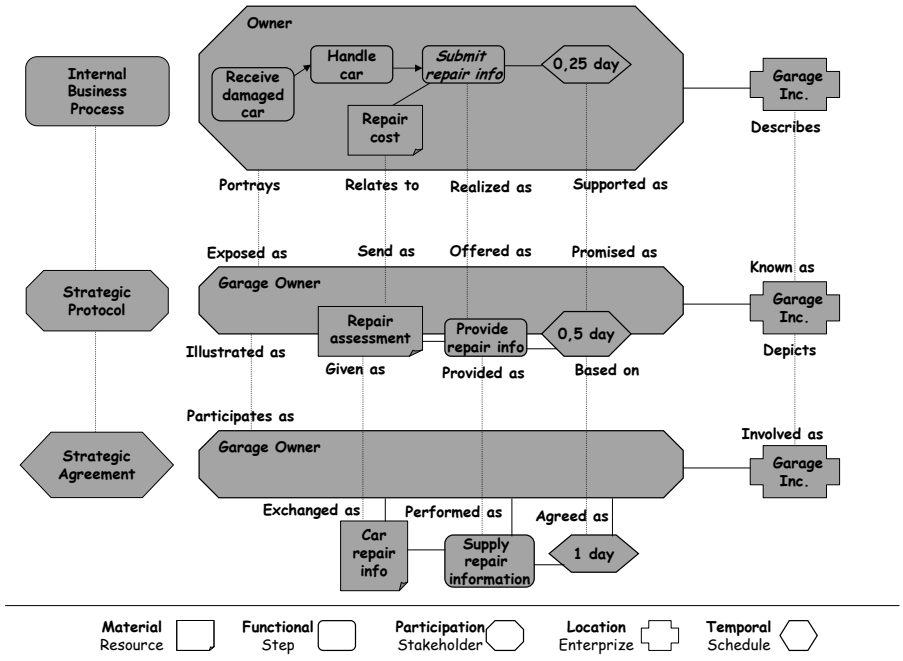

Figure 4.8: AGFIL Horizontal Mappings

consequences of a change in one business collaboration behavior can be traced to related behaviors at the same level (e.g. to assess the impact of an additional communication task in a private process on the associated protocol).

Specifically, we define five types of horizontal mapping categorized along part. These five mappings are discussed in detail in section 4.5.2 through 4.5.5 respectively; where the mentioned examples can be found in Fig. 4.8. We also discuss some illustrative dependencies that exist among quality and security requirements that are applicable to business processes, protocols and agreements respectively. Note that, as for vertical mappings, each horizontal mapping has two labels indicating the role played by each class in the relation.

4.5.1 Material Part

The mappings in material part make explicit what is communicated from an internal business process to the outside and vice versa. Internally consumed and produced resources, documents and messages are linked to those in a corresponding protocol via 'send as' and 'received as' relations. These relations are both one-to-one in nature. In the opposite direction each exposed resource, document and message also maps to exactly one resource, document, and message in a process. For example, `repair cost` will be send as `repair assessment` by `Garage Inc` in its interactions with others (as shown in Fig. 4.8). The exact constraints applicable to `repair cost` concerning quality, security, and etceteras must be more extensive or equal to that of `repair assessment`. If not, then `Garage`

Inc will be unable to fulfill its promises. Note that horizontal mapping of repair cost is only needed here since the involved step submit repair information is a communication activity. Also observe that depending on the exact type of communication the input and/or output of a communication activity will have to be mapped. For example, submit repair information of type 'supply' only concerns supplying repair cost and as such only this resource needs to be mapped to the protocol.

The connection from a protocol to an agreement is made in the material part by utilizing 'exchanged as' links. These express how exposed resources, document and messages are actually exchanged in the context of a specific business collaboration. In both directions 'exchanged as' links are one-to-one. To illustrate, repair assessment is exchanged as car repair info by Garage Inc when interacting with Lee C.S. Here as well there may be discrepancies between the exposed and agreed upon details of these two resources, for example that unauthorized disclosure of repair assessment is supported by Garage Inc but this is not required by Lee C.S. However, the stipulated conditions must never be more strict than offered promises. For example, suppose that Garage Inc has agreed in its business agreement that it will send car repair information in such a manner that it will not be disclosed to others. Then, in its business protocol Garage Inc must offer to exchange resource repair assessment in such a manner. Otherwise, the mapping between car repair information and repair assessment is not feasible.

4.5.2 Functional Part

To represent how the communication activities in an internal business process are exposed to the outside, they are mapped to corresponding activities in a protocol using 'offered as' relationships. Activities that do not entail communication with the outside will thus not be mapped using 'offered as ' relationships. For example, submit repair info is offered as provide repair info by Garage Inc. Note that the conditions applicable to submit repair info must be equal or more strict than those the exposed provide repair info (e.g. with regard to the level of quality required). If this is not the case, then Garage Inc is promising to provide repair information under conditions it can internally not support. Like the horizontal mappings for the material part, the cardinality of the offered relation is one-to-one at both ends (for those activities that constitute communication with the outside).

Garage Inc subsequently expresses how its potential communication activities are used within in its interaction with Lee C.S by mapping them via 'performed as' links to activities within their agreement. Each exposed step is linked to exactly one agreed upon step. Vice versa, every agreed upon step relates to exactly one exposed step. To exemplify, provide repair information is *performed as* supply car repair information by Garage Inc in its dealings with Lee C.S. Once more, the conditions applicable to supply car repair information must not be stricter than those of provide repair information as this would imply that Garage Inc has committed itself to an agreement it can not fulfill. To illustrate, if Garage Inc has promised that provide repair information will be fully accessible between the working hours of 9am to 5pm, then it must not be the case that

the corresponding private step `submit repair information` is not fully accessible during that same period. If so, then the mapping is incorrect.

4.5.3 Participation Part

The purpose of horizontal mappings in the participation part is to make explicit who is actually involved in a business collaboration across the different types of business collaboration behavior. The participation part exponents in these different behaviors, i.e. stake holders, actors and services, are linked via 'portrayed as' and 'participates as' relationships. The former connect stake holders, actors and services in an internal business process responsible for performing communication activities to those in a protocol. The latter associate the stake holders, actors and services in a protocol to those in an agreement. The 'portrayed as' relation has multiplicity one-to-one at both of its ends just like the 'participates as' relation.

Examples of horizontal mappings can be found in Fig. 4.8, where `owner` of `Garage Inc` is portrayed as `garage owner` to the outside, and subsequently participates as `garage owner` in the AGFIL business collaboration. The amount and type of information defined about the garage owner internally will vary from what is exposed to the outside. For example, private details will not be disclosed whereas contact information is likely to be provided. A similar decrease in provided information may be expected when going from `garage owner` in the protocol to the corresponding stake holder in the agreement. Here it is not possible that `garage owner` in the latter contains more details than the former.

4.5.4 Location Part

In the location part the reason to explicitly define horizontal mappings is to be able to trace which enterprizes, units and endpoints are participating in the business collaboration. Enterprizes, units and endpoints are linked from one type of business collaboration behavior to the other by defining 'known as' and 'involved as' relations between an internal business process and protocol, and a protocol and agreement respectively. The numeric constraints on these relations are the same as those applicable to the horizontal relations of the other parts.

The example in Fig. 4.8 demonstrates these mappings for `Garage Inc`, conveying that `Garage` is known to others as `Garage Inc`. It also operates under this name in the context of the AGFIL business collaboration. Important to realize in this regard is that the information known about `Garage Inc` as an organization internally will often be much more extensive than what is exposed to other parties and/or present in an agreement. Thus, an information discrepancy similar to the one found in the participation part will likely exist for horizontal mappings in the location part.

4.5.5 Temporal Part

The motivation underlying the usage of horizontal mappings in the temporal part is to convey how internal temporal requirements are related to offered temporal conditions and how these in turn correspond to agreements concerning the factor 'time' in a business collaboration. Schedules, events and triggers in an internal business process are mapped to a protocol using 'published as' and 'accepted as' links with cardinalities identical as discussed for those relating the other parts in these aspects. Fig. 4.8 provides an example where the schedule for provision of `repair information` is internally `0,25 days` and published as `0,5 days` (as such `Garage Inc` attempts to ensure that it will always meet this schedule, since internally provision of car repair information could be done much quicker).

In turn, 'agreed as' links are employed to relate modeling elements representing the temporal part in a protocol and agreement respectively. To exemplify, the schedule `0,5 days` of `Garage Inc` is agreed to by `Lee C.S` as `1,0 days`, i.e. somewhat less strict. Observe that the reverse should not be possible. That is, it must not be the case that `Garage Inc` agrees to provide `Lee C.S` with car repair information within half a day, whereas it is only capable of doing so in one day. In other words, the conditions applicable to the exponents of the temporal part in an agreement must not be more strict than those in the corresponding protocol.

4.6 Business Collaboration Information Model

In the previous subsections we introduced a wide variety of models and mappings to capture the business collaboration context provided by the BCCF. To avoid running the risk of ending up with a plethora of model languages, we advocate the need for a single language to express all these different models and mappings in an uniform manner. This language is to be expressive enough to cover everything discussed so far, yet be simple in its basic structure. Moreover, the language must provide underpinnings for the different models and mappings in order to allow the verification of business collaboration designs. To meet these requirements we have developed the *Business Collaboration Information Model (BCIM)*. The BCIM constitutes a set of so-called *modeling description atoms* that constitute the basic building blocks with which we construct models and mappings to capture the different business collaboration behaviors in Fig. 4.2 resulting in a design of a business collaboration. An overview of the BCIM is provided in Fig. 4.9.

The five bottom entities in the figure represent the five types of modeling description atom: *context*, *element*, *property*, *link* and *attribution*. The first four types of atom are used to form a model, whereas the last type of atom is used to construct a *mapping*. Models and mappings themselves are contained within a *design*, which describes the entire context of individual business collaborations. The modeling description atoms, models and mappings, and subsequently designs are defined using a blend of a first order logic (FOL) and set theory like notation, and serve the following purpose:

1. *context*: identifies the position of a model within the business collaboration context.

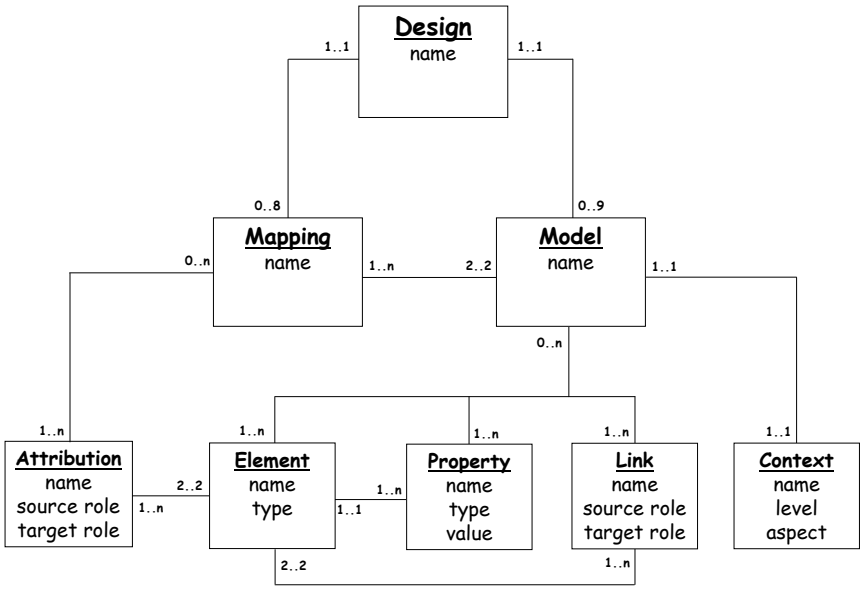

Figure 4.9: Business Collaboration Information Model (BCIM)

A context depicts a level and an aspect. Level must be equal to 'strategic', 'operational' or 'service', and aspect equal to 'internal business process', 'protocol' or 'agreement'). Formally this is represented as:

Definition 1

A context **c** is formally defined as a tuple C(cn,cl,ca,cm); where 'cn' functions as identifier, 'cl' is the level, 'ca' the aspect, and 'cm' identifies the model to which the context belongs.

□

For example, `C(strategic,agreement,AGFIL-STM)` represents the context of the model `AGFIL-STM` which describes the strategic agreement made between `Garage Inc` and `Lee C.S`.

2. *element*: represents a part of a business collaboration behavior, i.e. material, functional, participation, location, or temporal part. An element has an uniquely identifying name and a type. The element type reflects the kind of part being represented at particular level. Each element has one or more properties. The formal definition is:

Definition 2

> An element **e** is formally defined as E(en,et,em); where 'en' is the name of the element, 'et' the type, and 'em' the model reference.

□

To illustrate, `E(supplyRepairInformation,step,AGFIL-STM)` expresses the `supply repair information` step performed by `Garage Inc` in `AGFIL-STM`.

3. *property*: defines a characteristic of an element, enriching the description of a part. Each property has a name, type and value. The name provides an unique identifier, whereas the type reflects the kind of characteristic being defined (which can be both basic and advanced in nature); and value defines the value of the property. Formally this gives:

Definition 3

> A property **p** is formally defined as P(pn,pt,pv,pe,pm); where 'pn' is the name of the property, 'pt' the type, 'pv' the value, 'pe' the element reference, and 'pm' the model reference.

□

To exemplify, in order to indicate that the price of operation `manage claim` is equal to $500, the property `P(myProp,price,500,manageClaim,AGFIL-SEM)` can be specified.

4. *link*: expresses connections between elements belonging to the same model. Links have a name, source element and role, and target element and role. The name is for identification purposes, whereas the source and target role indicate the kind of relationship being established. The formal definition is:

Definition 4

> A link **l** is formally defined as L(ln,lso,ls,lta,lt,lm); where 'ln' is the name of the link, 'lso' the source element, 'ls' the role of the source element, 'lta' the target element, 'lt' the role of the target element and 'lm' the model reference.

□

For example, `L(myLink,garageRepairer,responsibleFor, supplyRepairInformation,allocatedTo,AGFIL-STM)` conveys that `garage repairer` is responsible for performing `supply repair information` (or vice versa that this task has been allocated to `garage repairer`, as stipulated in the strategic agreement model `AGFIL-STM`.

5. *attribution*: specifies relations between elements from different models, i.e. express mappings among elements. An attribution has a name, source element and role, and target element and role. The name gives an unique label to the attribution; the source and target role signify the kind of attribution defined between the attribution's source and target element. An attribution can be 'vertical' in nature linking elements from models at different levels, or 'horizontal' connecting elements from models at different behaviors. This formally amounts to:

   Definition 5

   An attribution **a** is formally defined as A(an,aso,as,ata,at,am) where 'an' is the name of the attribution, 'aso' the source element, 'as' the role of the source element, 'ata' the target element, 'at' the role of the target element, and 'am' the name of the mapping to which the attribution belongs.

□

   Attribution `A(at,carRepairInformation,leadsTo,carRepairReport, resultOf,AGFIL-MAP)` defines such an attribution stating that `car repair information` leads to `car repair report` in the mapping `AGFIL-MAP`; or alternatively interpreted as that `car repair report` is the result of `car repair information`.

6. *model*: represents a particular model e.g. the `AGFIL-STM`. A model has a context, and constitutes one or more elements, properties and links. Formally this gives:

   Definition 6

   A model **M** is formally defined as $M = (c, e, p, l, \mid e \in ES \wedge p \in PS \wedge l \in Lc.cm = e.em \wedge e.em = p.pm \wedge p.pm = l.lm)$

   where

   ES: a set of elements defined as $\{e_0...e_n\}$.
   PS: a set of properties defined as $\{p_0...p_n\}$.
   LS: a set of links defined as $\{l_0...l_n\}$.

□

7. *mapping*: defines a mapping between two models such as the agreements in `AGFIL-STM` and `AGFIL-OPM`. A mapping has a name, and consists of a collection of attributions. The formal representation is:

Definition 7

A mapping **MAP** is formally defined as MAP$\{_{MS1\text{-}MS2}\}$ = (ms$_i$, ms$_j$, a | ms$_i$∈ MS$_1$ ∧ ms$_j$ ∈ MS$_2$ ∧ a∈ A ∧ a.aso=ms$_i$.e ∧ a.ata=ms$_j$.e)

where

AS: a set of attributions defined as {a$_0$...a$_n$}.
MS: a set of models defined as {M$_0$...M$_n$}.

□

8. *design*: describes an individual business collaboration as a collection of models and mappings. Formally it is defined as:

Definition 8

A design **D** is formally defined as $D = (ms, mp, \mid ms \in MS \wedge mp \in MAPS \wedge mp \in MAPSmp.ms = ms)$

where

MS: a set of models defined as
{M$_{stp}$,M$_{opp}$,M$_{sep}$,M$_{stpr}$,M$_{oppr}$,M$_{sepr}$,M$_{sta}$,M$_{opa}$,M$_{sea}$}

where M$_{stp}$, M$_{opp}$, and M$_{sep}$ are the strategic, operational and service process respectively; M$_{stpr}$, M$_{oppr}$, and M$_{sepr}$ the strategic, operational and service protocol respectively; and M$_{sta}$,M$_{opa}$, and M$_{sea}$ the strategic, operational and service agreement respectively.

MAPS: a set of mappings defined as
{MAP$_{stp-opp}$,MAP$_{opp-sep}$,MAP$_{stpr-oppr}$,MAP$_{oppr-sepr}$,MAP$_{sta-opa}$,
MAP$_{opa-sea}$,MAP$_{stp-stpr}$,MAP$_{opp-oppr}$,MAP$_{sep-sepr}$,MAP$_{stpr-sta}$,
MAP$_{oppr-opa}$,MAP$_{sepr-sea}$}

where MAP$_{stp-opp}$ and MAP$_{opp-sep}$ are the mappings between the strategic and operational, and operational and service process respectively; MAP$_{stpr-oppr}$ and MAP$_{oppr-sepr}$ the mappings between the strategic and operational, and operational and service protocols respectively; MAP$_{sta-opa}$ and MAP$_{opa-sea}$ the mappings between the strategic and operational, and operational and service agreement respectively; MAP$_{stp-stpr}$, MAP$_{opp-oppr}$,

and $MAP_{sep-sepr}$ the mappings between the strategic process and protocol, operational process and protocol, and service process and protocol respectively; and $MAP_{stpr-sta}$, $MAP_{oppr-opa}$, and $MAP_{sepr-sea}$ the mappings between the strategic protocol and agreement, operational protocol and agreement, and service protocol and agreement respectively.

□

As the last two definitions show we are able to represent all the different models and the dependencies between them (as discussed in sections 4.1 through 4.5) by using the generic constructs of context, element, property, link and attribution. We can also define any advanced requirements (like the ones for quality and security) through the addition of appropriate properties to elements. As such, due to its generic nature the BCIM provides a very rich and expressive modelling language to capture the context of business collaborations as defined in the BCCF in Chapter 3 in designs. In the next section we will review the qualities of the BCIM and the model based approach in general in light of the related work in this area and the research objectives identified in section 1.4 of Chapter 1.

4.7 Discussion

In this chapter we discussed a model based approach to enable organizations to capture their business collaborations in the form of models. Throughout this discussion we made occasional reference already to comparable models in existing literature on business process and collaboration modeling. The usage of models to capture strategic requirements is for example advocated by (Bresciani et al., 2004), (Traverso et al., 2004) and (Yu, 1997). The resulting models are by and large similar to the strategic models we describe in terms of the concepts that they use. Compared to (Gordijn et al., 2006) we do not provide constructs to make value exchanges explicit, as we consider this to be in the realm of economic considerations and as such outside the scope of this research. However, as (Gordijn et al., 2006) shows it is possible to fairly easily relate value proposition models to strategic models more of the form of i∗ models in (Yu, 1997); which are very much akin to our strategic models as they also provide the means to describe resource exchanges between organizations.

Many proposals have also been made to model (parts of) the operational level including but not limited to business process languages like (Business Process Modeling Initiative, 2002) and (Business Process Modeling Initiative, 2003), collaboration specifications such as (ebXML Initiative, 2006) and (ebXML Initiative, 2002), workflow oriented proposals e.g. proposed in (van der Aalst et al., 2003), (Georgakopoulos et al., 1995), (Mentzas et al., 2001), (Workflow Management Coalition, 2002), (Casati et al., 2000) and (Workflow Management Coalition, 1995), role and agent based approaches such as (Phalpa et al., 1998) and (Dubray, 2003), and (Jennings et al., 1996) and(OBrien and Wiegand, 1998), event-oriented works like (Rittgen, 2000), (Loos and Fettke, 2001) and (Mendling and Nttgens, 2004), data-oriented research including the older OO (Meyer, 2000; Booch et al., 1998) and more recently semantic web based attempts such as (DAML Services Coalition, 2003),

(Cardoso and Sheth, 2003) and (Laukkanen and Helin, 2003). At service level the list is almost equally long encompassing (Christensen et al., 2001), (Curbera et al., 2002), (Banerji et al., 2004), (Bussler, 2001) (Fensel and Bussler, 2002), (Casati et al., 2003), (Dijkman and Dumas, 2004) and so on. More formally oriented works include process algebras like pi-calculus (Milner, 1990) and (Milner, 1993) and CSP (Hoare, 1985), petri nets (van der Aalst, 1998) and (Narayanan and McIlraith, 2003), and simple finite-state automata.

The main concern we have with these proposals is that they tend to focus on modeling one type of business collaboration behavior. They typically allow the definition of private processes, protocols or agreements in the BCCF. For example, the solutions reported in (Workflow Management Coalition, 1995) and (Business Process Modeling Initiative, 2002) are intended for defining internal business process models at operational level. Other works are intended to be used in conjunction with others but they then lack a common underlying language. To illustrate, the proposals in (Curbera et al., 2002), (Christensen et al., 2001), (Little et al., 2005) and (Banerji et al., 2004) could in theory be used in combination to capture internal business processes, protocols and agreements at service level respectively. However, they are not expressed in exactly the same terms. As a result it is often not possible to make dependencies among them explicit as can be done in our approach. Other works like (Bresciani et al., 2004) offer this opportunity but choose not to, do not give any details on how to do it for example (ebXML Initiative, 2002), or support it but only at a particular level such as (Traverso et al., 2004).

Another issue is that the mapping between models at different levels is often neglected. Most of the works discussed in section 2.2 of Chapter 2 limit themselves to a single level not taking their relation to other levels into consideration. A notable exception includes (Bresciani et al., 2004), however, this approach has the disadvantage that not all parts in the business collaboration context are covered in their models. An interesting idea is presented in (Veryard, 2003) where the suggestion is made to express business and IT pervasively in terms of services resulting in business and technical services respectively. It is unclear however what this translates to at strategic level. Similar is Archimate (Jonkers et al., 2003) whose mappings also center around service orientation. However, it also does not cover the strategic level. Another proposal for a service-oriented modeling architecture (SOMA) is under development by IBM (IBM, 2006), but as of yet it has not been made public yet. Note by the way that although we do not adopt a pervasive service view of business collaborations our modeling approach could be easily adapted to accommodate this. In such approach tasks and steps at operational and strategic level respectively would become the equivalent of operations in services at technical level for so-called operational and strategic services. For now we prefer not to do so, as we feel that is more alien to developers accustomed to currently used modeling solutions like workflows and i$*$ models.

With regard to the dependencies that exist among the different aspects of business collaboration (i.e. private business processes, business protocols and business agreements), several solutions have been developed as well. (Peltz, 2003) discusses orchestration versus choreography at service level to describe service based private and public processes respectively. (Traverso et al., 2004) informally identifies the issues involved when reconciling global and local requirements of organizations. However, no mappings are made explicit.

(Dijkman and Dumas, 2004) does explore this topic providing formal underpinnings for the different aspects introduced in the BCCF (and additionally provider behavior) as well as the interrelations among these aspects. Their approach is thoroughly worked out, but is unfortunately limited to the service level. As such, its application on strategic and operational level remains of yet an open issue. The work in (Bussler, 2001) offers intriguing notions in relation to business-to-business protocols, where they develop representations for dependencies between interface processes (i.e. protocols) and private business processes through so-called binding processes. Particular emphasis is placed on capturing the flow of messages to and from an organization as they move from interface process to private process and back. This resembles our proposal for mapping messages in protocols to those in processes. However, we also define such mappings for the other parts of the protocols and processes. Moreover, in the presented model based approach similar mappings can also be specified at operational and strategic level.

Recapping the above we feel that one of the major contributions of the model based approach is that it allows organizations to model a wide variety of requirements in different types of business collaboration behavior at different layers of abstraction taking different parts into account. These models can be defined in terms of constructs with clear semantics as defined in a single modeling language, the BCIM, something that is not possible with most existing proposals. In addition, we provide the means with which organizations can make explicit dependencies among models describing different aspects. We also enable organizations to depict how their strategic, operational and service requirements are related. Work in the latter area has been relatively scarce so far, and we therefore consider this a second important contribution to the business collaboration development and management research area. Finally, the third major accomplishment lies in the definition of a generic Business Collaboration Information Model to uniformly express all the different models proposed for business collaboration in literature. As such, with the BCIM organizations have a very rich language for business collaboration modeling at their disposal. Moreover, as we will demonstrate in Chapter 5 and 6 of this dissertation it allows us to define a rule based approach with which organizations can develop and manage their business collaborations in a dynamic manner. The main weakness of the model based approach in our view is that it currently provides little support for the specification of advanced requirements pertaining to issues like quality and security. Although this is conform the scope set in section 1.5 of Chapter 1, more support is needed for capturing such requirements if organizations are to be able to accurately describe their business collaborations. However, we did demonstrate how such specification can be done by introducing appropriate properties extending the different classes in the different meta-models. As such, the proposed model based approach can be extended in a straightforward manner to provide the required support.

Linking back to the third research question we established in section 1.6 of Chapter 1 concerning the representation of business collaborations, the model based approach answers this question as follows:

1. Firstly, we developed different models for the different aspects at the different levels

within the BCCF. This empowers organizations to capture their business processes, protocols and agreements whilst taking both business and technical requirements into account covering material, functional, participant, location and temporal parts. We furthermore placed all the models in relation to existing works in business collaboration development (where possible) to ensure that they are similar to current standards adopted by organizations in order to ease their adoption.

2. Secondly, with the different models in place we then made explicit the dependencies between aspects in terms of horizontal mappings. The capacity to define such horizontal mappings enables organizations to change their processes, protocols and agreements whilst at the same time maintain their consistency. Moreover, it provides organizations with a way of explicitly relating collaborations with one partner to those with another in case such interdependencies exist. This is of vital importance for organizations as they often have to operate in and manage complex value webs and supply chains.

3. Thirdly, we explained how through the usage of vertical mappings organizations can make explicit how strategic behavior is realized in terms of daily operational routines, and subsequently how these routines are facilitated for by their IT-infrastructure. As such, organizations can visualize how business is aligned with IT making this alignment accessible and transparent. This can be done for their private business processes, their published business protocols as well as their made business agreements.

4. Fourthly, we developed a highly expressive, generic Business Collaboration Information Model (BCIM) to capture all the different models and mappings needed to describe the context of business collaboration in terms of a small set of modeling description atoms. Moreover, the BCIM provides underpinnings for these modeling description atoms and thus for business collaboration modeling. As such, with the BCIM organizations have the means to design their business collaborations in an uniform and unambiguous manner.

These four points illustrate that we have accomplished the goal in this chapter of providing organizations with the means to capture their different business collaboration requirements. The next step, as reflected in the research objectives established in section 1.4 of Chapter 1, is to investigate how the development and management of business collaboration can be done in a dynamic yet consistent manner. The solution we propose on the basis of rules, is the topic of the next chapter.

Chapter 5

Rules In Business Collaboration

Rules are for the obedience of fools and the guidance of wise men; Douglas Bader

Rules. We don't need no stinking rules; Adapted from Bernard Traven

In Chapter 4 we discussed how organizations can utilize a model based approach to capture the context in which their business collaborations take place. In this discussion we covered a wide range of models, all of which describe a specific part of a business collaboration. We also developed a generic Business Collaboration Information Model with which these different business collaborations models can be defined in an uniform manner. This brings us to the third step in the research road map (see also Fig. 5.1), being the identification and specification of rules for business collaborations. As can be seen in the figure this will result in a classification of the rules of relevance for business collaboration as well as a generic *Business Collaboration Rule Language* with which the different rules can be uniformly described.

Of course the question begs as to why the usage of rules is desirable for the development and management of business collaborations. In a nutshell the problem is as follows: as the discussion of the model based approach in Chapter 4 has shown organizations need to carry out an extensive modeling effort in order to fully capture all the specifics of their business collaborations. Organizations are faced with the challenge of capturing their processes, protocols and agreements at three layers of abstraction each of which covers five different parts augmented optionally with various advanced domains such as quality and security. Even when we take the benefits into account of modeling business collaborations in a modularized manner using a generic Business Collaboration Information Model, this still leaves organizations with the task of creating and maintaining all the different models and the dependencies between them whilst at the same time ensuring that individual models are consistent, models at different levels are aligned, and models capturing different aspect do not contradict one another. In addition, organizations must ensure that the modeled collaborations are and remain compliant with the requirements applicable to them. However, the problem is that such requirements are frequently subject to change and consequently the model(s) capturing affected behavior(s) will often have to

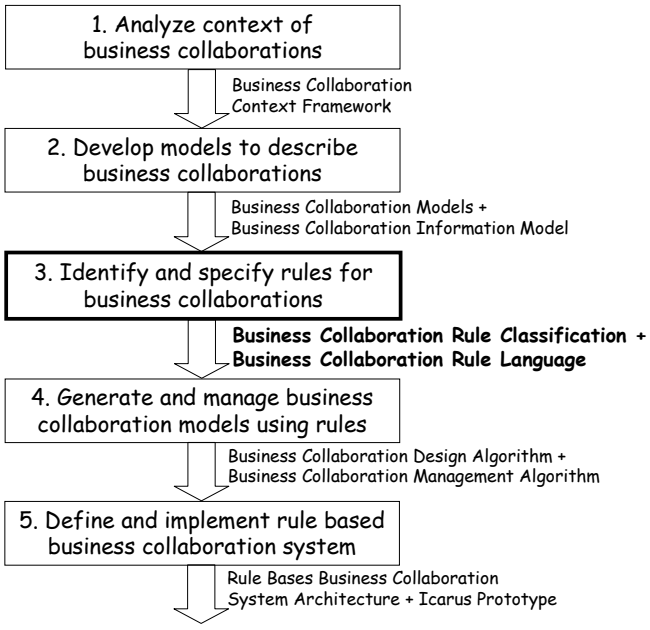

Figure 5.1: Research Road Map - Classifying And Specifying Rules

be adjusted.

To illustrate, suppose that at the service level `Lee C.S` develops a new security policy for `claim management service`, which adopts an updated messaging protocol because of new technical regulations. However, because of this the service can no longer communicate with `car repair service` of `Garage Inc`. This thus leads to the violation of the technical agreement between `Lee C.S` and `Garage Inc`. Because `report estimate` has become unavailable, `get estimate` at operational level can no longer be successfully completed by `consultant`. This then results in the violation of the operational agreement between the parties, which consequently can jeopardize their high level strategic objectives (as defined in the AGFIL-STM). As even this simple example already shows a change in one part of a business collaboration can have an enormous, cascading affect on the collaboration as a whole. Therefore, given the complexity and sheer scope of the modeling effort it is simply not feasible in our view to perform business collaboration development and management manually whilst at the same time be able to do this when requirements are often changing. Thus, organizations require some mechanism to give them the ability to cope with change in an easy and effective manner.

In this and the next chapter we present a rule based approach to facilitate dynamic business collaboration development and management whilst ensuring their consistency. Rules are basically statements that tell you what to do or not to do. Organizations use rules to guide and control their activities in order to conduct business in a consistent and desired manner. Rules have been around in one form or another in organizations since

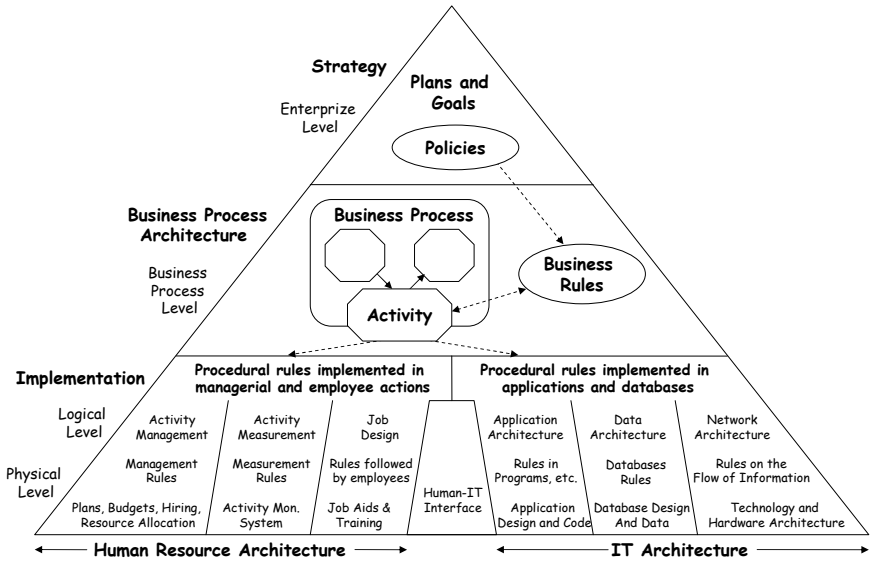

Figure 5.2: Role Of Rules In Enterprizes

they first came into existence, be it implicit heuristics applied by individual employees, working instructions explicitly stipulated in employee work manuals, legislation imposed by the government, or decision logic hard coded into databases as integrity constraints or into applications as if-then statements. Almost as long as there have been rules, there have also been rule approaches. Before the advent of IT these were typically paper based of which manuals, contracts, and so on, are examples. As computers gained popularity many attempts have been made to employ rules in a computational setting. Early exponents in the 1960s were expert systems, which focused on making explicit the tacit knowledge of experts such as doctors, lawyers, and so on. More recently the so-called *business rules approach* has been promoted, for example in (Date, 2000), (von Halle, 2002), (Moriarty, 1993), (Ross, 1997) and (Ross, 2003). This approach is a development methodology where rules are in a form that is used by, but not embedded in business process enactment systems, to improve the dynamicity of the managed processes i.e. the ease with which new and existing processes can be changed. What these approaches have in common is that they are all centered around the notion of taking rules and making them explicit, as such making them manageable. Within organizations there is an abundance of such rules as Fig. 5.2 illustrates.

At the top of the pyramid displayed in the figure we find the overall plans and goals of organizations. These usually consist of vague, high level descriptions of the organization's objectives, for example maximizing of the profit, being innovative and so on. To achieve these objectives they are depicted somewhat more formally in business policies.

Such policies describe how the business of the organization should be conducted in order to realize the overall goals and objectives. Policies are often written down in handbooks, for example a sales policy handbook that provides a description of how a customer order is to be processed. Business policies are implemented in business rules, which provide concrete statements that enforce constraints on the business processes to achieve the targeted objectives. For their enforcement they depend on rule implementations on the logical and (subsequently) physical level. Two main types of rule can be distinguished on these levels: human resource related and IT-related rules. Human resource related rules are concerned with governing the 'soft' part of the organization. That is, they drive the part of the organization, which is not implemented in some IT-technology. In contrast, IT-related rules are encoded in one form or another into applications, databases, networks and other sorts of IT technologies. When placed in the context of business collaboration, the picture in Fig. 5.2 becomes even more complex. As organizations collaborate they will each have their own plans and goals, business rules, and implementation level rules. All these different rules will somehow have to be consistent with each other in order for collaborations between organizations to progress in a compliant and consistent manner.

The problem that many organizations currently face is that their automated business processes have business logic embedded inside. As such, they often take substantial time to change, where such changes can be prone to errors. In the current dynamic business environment the life span of business models has been greatly shortened, and as such it is critical for organizations to be able to adapt to changes in a rapid manner. Adopting a rule approach achieves this by adhering to four fundamental principles (the so-called STEP principles identified in (von Halle, 2002)):

- **Separate rules**

  Instead of embedding rules in code, writing them down in manuals no-one reads, and so on, rules are developed and managed separately in a rule approach. By placing them into *rule management systems (RMS's)*, which support the authoring, deployment and management of rules, these rules become more accessible. Rules also become available for reuse as organizations can take existing rules and apply them in other circumstances. Furthermore, separation facilitates rule consistency checking thus assisting organizations to determine whether rules do not lead to conflicting situations, e.g. between high level goals and business rules or between the implementation level rules of two individual organizations.

- **Trace rules**

  Explicitly separating rules also increases their traceability. That is, business people can see where a rule comes from and why it is applied in the way it is. On the one hand this greatly enhances the capacity of organizations to explain and reason about their motivation, that is, why they do things the way they do. On the other hand it provides organizations with the capability to assess the impact of rule changes on the business, for example to determine the affect of a high level policy change on the operational business processes, or on its capability to cooperate with others.

- **Externalize rules**

  Externalization of rules makes organizations aware of the fact that there exist rules and that these rules govern the business. Moreover, as rules are expressed in natural (or semi-natural) languages they can be more easily understood. This also means that the business can now manage its own rules rather than having to depend on the IT department. In addition, knowing what rules exist and being able to find out what these rules exactly are, enables organizations them to challenge the rules for example while negotiating the terms of a business collaboration with other parties.

- **Position rules for change**

  If the rules are successfully challenged, then they should be changed. Separating and externalizing rules enables organizations to position them in such a manner that modification can be done in an easy and quick manner. This effectively gives them the capability to manage their strategies, business rules and implementation level rules in a dynamic manner with which changes can be easily effectuated from strategies to processes to IT architecture and vice versa. It also provides them with the ability to assess the impact of changes to their private processes on their cooperations with others (and vice versa). This gives organizations extensive control over their business collaborations, where they can easily make desired changes.

Thus, the crux of a rule based approach is that by making rules explicit they become manageable. In the first section, section 5.1, we will introduce an approach for dynamic business collaboration that is based on this notion, where rules are used to govern and control the design of collaborations. The remainder of the chapter is then dedicated to exploring the concept of rules in more detail, in particular in relation to business collaboration. We begin by defining what we actually mean when we are talking about rules for business collaborations in section 5.2. Then, in section 5.3 we develop a classification of rules for business collaboration development and management, where the aim is to identify what types of rule need to be present to drive and constrain the development and management of business collaboration designs. Next, we discuss the Business Collaboration Rule Language (BCRL) to facilitate the specification of rules in section 5.4. Finally, we compare the obtained results in section 5.2 to 5.3 to the existing rule literature in section 5.5. We also evaluate the merits of these results in relation to the research objectives stipulated in section 1.4 of Chapter 1. To exemplify the ideas presented we provide illustrative examples from the AGFIL case study throughout the chapter.

5.1 Using Rules

As just observed, the general idea behind a rule based approach is that the manageability of rules is increased when they are explicitly specified. We apply this notion in the context of business collaborations where the idea is to separate the requirements that govern business collaboration from the actual designs, express these requirements as rules and then use

these rules to drive the design of business collaborations. Concretely, in this dissertation we advocate a style of development in which designs are built from the different BCIM modeling description atoms in a rule based manner. Specifically, we associate each BCIM element with a set of rules called a *policy*. The reason that we choose elements here is that they are the most fine-grained and independent concepts in the BCIM with real life semantics. Properties, links and attributions also have such semantics but exist only by the grace of elements; and as such they can be derived based on the existence of elements. Contexts are independent but these constructs are used to reflect the position of a model within the business collaboration context rather than expressing real life semantics. Models and designs do possess real life semantics yet they are more coarse (since they comprise of multiple BCIM elements), which results in a lesser degree of dynamicity; as policies can then only be adjusted at the model or design level rather than per individual BCIM element.

Having said that, a BCIM element policy comprises of two types of rules: firstly, so-called *derivation rules* define how the particulars about a business collaboration can be deduced. These derivation rules are used to determine the values of a BCIM element's properties, and the links and attributions it will have with other BCIM elements. An example of such rule for `Garage Inc` is that after `estimate repair` has been completed `repair car` must be performed if the estimate was below $500. Secondly, whereas derivation rules state how BCIM elements should be defined and combined, so-called *control rules* depict which are valid definitions and combinations of BCIM elements and which are not. Specifically, control rules constrain the property values, and links and attributions of BCIM elements to ensure that these elements are defined and combined in a consistent manner to form business collaboration designs. To illustrate, the rule of `Garage Inc` that the maximum time between receipt of a car and completion of the repair is 10 days (thus linking the dates of two events) is a control rule. Summarizing the previous, we have pre-defined BCIM elements and their policies that function as the basic building blocks for business collaboration designs; where each policy constitutes a set of derivation and control rules.

Then, to facilitate development of the different parts of business collaborations conform the BCCF, we say that each model in a business collaboration design comprises a collection of BCIM elements and policies that together form the building blocks of the model describing a business collaboration behavior. We refer to such collection as a *model schema*. Formally we define a model schema as:

Definition 9

> A schema $\mathbf{S}_m$ for a model M is formally defined as S_m=$\{eP_0...eP_n\}$; where each 'eP' constitutes the combination of a BCIM element 'e' and its policy P. A model M based on S_m is said to be an interpretation of S_m, where multiple interpretations can be based on the same S_m.

□

As a business collaboration design comprises of multiple models we can then state that the schema for this design, called a *design schema*, is a set of model schemas. This is formally defined as:

Definition 10

> A schema $\mathbf{S}_d$ for a design D is formally defined as S_d={S_{stp},S_{opp},S_{sep},S_{stpr},S_{oppr},S_{sepr}, S_{sta},S_{opa},S_{sea}}. Here S_{stp}, S_{opp} and S_{sep} are the model schemas of the strategic, operational, and service process respectively; S_{stpr}, S_{oppr} and S_{sepr} are the model schemas of the strategic, operational, and service protocol respectively; and S_{sta}, S_{opa} and S_{sea} are the model schemas of the strategic, operational, and service agreement respectively. A design D based on S_d is said to be an interpretation of S_d, where multiple interpretations can be based on the same S_d.

□

Note that the above definition effectively means that S_d can be regarded as well as a collection of BCIM elements and their policies, i.e. {eP_0...eP_n}, which is the result of the conjunction of the separate sub-collections comprised in the different model schemas. As we will show in section 6.2.2 of Chapter 6 this definition of a design schema is useful when organizations are concerned with the generating of designs for their business collaborations. In contrast, we will see in section 6.1 that when organizations are developing or modifying design schemas for these collaborations it is easier to adopt Def. 10. Also observe that if all the rules in the policies of the BCIM elements in S_d have no conditions (i.e. conveying facts), then S_d has only one possible design D as its interpretation. Such S_d is the equivalent in the rule based approach of a completely static business collaboration design in which nothing is variable.

Now, in the business collaboration between `Garage Inc` and `Lee C.S` there will be two design schemas S_d as defined in Def. 10 belonging to these respective organizations. Conform the definition of a design as a collection of models across three levels and aspects (see Def. 8 in section 4.6 of Chapter 4) each design schema will have model schemas defining the building blocks of the internal processes, public protocols and agreements at the different levels. That is, each design schema has: 1) process schemas S_{stp},S_{opp} and S_{sep}; 2) protocol schemas S_{stpr},S_{oppr} and S_{sepr}; and 3) agreement schemas S_{sta},S_{opa} and S_{sea}. Because of the separation of a design schema into the different model schemas, it becomes possible to develop each model schema on an individual basis. Also, it allows organizations to reuse model schemas across different design schemas. For example, the private processes of `Lee C.S` support both the interactions with `Garage Inc` as well as `AGFIL`. To express this `Lee C.`S can share the model schemas underlying these processes across its design schemas for its interactions with these two parties. Such reuse can be accommodated for example for design schemas by include model schemas through name based references. Moreover, by sharing the model schemas underlying their agreements

`Garage Inc` and `Lee C.S` can incorporate the expected behavior of each party in their respective design schemas.

Based on the previous the development of business collaborations by individual organizations like `Garage Inc` and `Lee C.S` becomes a matter of defining design schemas by creating the required model schemas. We envision that first both organizations will define the different model schemas for their private processes and protocols at strategic, operational and service level in such manner that these are consistent. Subsequently, when `Garage Inc` and `Lee C.S` wish to collaborate, they can extend their design schemas with model schemas that capture the conditions under which they have agreed to collaborate. If the model schemas underlying the protocols of `Garage Inc` and `Lee C.S` are consistent with each other, then it will be possible to form such agreements. However, if this is not the case but the two organization are still intent on working together, then changes will be required to their respective protocol schemas. In such case both `Garage Inc` and `Lee C.S` will assess whether these changes consequently affect the model schemas underlying their private processes (Note: a reverse style of development is possible as well, that is, `Garage Inc` and `Lee C.S` can first define a model schema for an agreement or use a predefined agreement schema and then (re-)define the model schemas underlying their private processes and public protocols as necessary.).

If changes to the model schemas underlying private processes are indeed required and these schemas are shared across different design schemas, then any changes to them can potentially affect multiple business collaborations. For example, if `Lee C.S` has to adjust the model schema of its operational process for `accountant` and assuming that this model schema also captures the process underlying its interactions with `AGFIL` (i.e. the model schema is shared across the design schemas underlying `Lee C.S`'s interactions with both `Garage Inc` and `AGFIL`), then this can result in modification of the agreement that `Lee C.S` has with `AGFIL`. Concretely, if the model schema for the protocol of `accountant` regarding its dealings with `AGFIL` is no longer consistent with that of the modified process schema, then `Lee C.S` will have to adjust its the protocol schema. This in turn can lead to inconsistencies between this protocol schema and the agreement schema `Lee C.S` shares with `AGFIL`. Thus, as a result of a change induced by `Garage Inc` the business collaboration between `Lee C.S` and `AGFIL` will need to be adjusted. In this sense the sharing of model schemas across multiple design schemas allows organizations like `Lee C.S` to trace the impact of changes from one business collaboration to another (like we described in section 3.1.4 of Chapter 3).

Now, as we observed a design schema can have multiple interpretations. That is, multiple designs can be created based on a single design schema. As such, the exact interpretation (i.e. design) that will be defined, depends on the circumstances of the specific business collaboration that is to be designed. Because such information is to a large extent dependent on what happens as a business collaboration is running, the consequence is that we can only determine how a design must look at runtime. To accommodate this we combine the above introduced manner of developing business collaborations with the idea of dynamically creating designs at runtime. That is, a business collaboration design shapes itself to the circumstances of the collaboration as it is progressing. There is thus no pre-

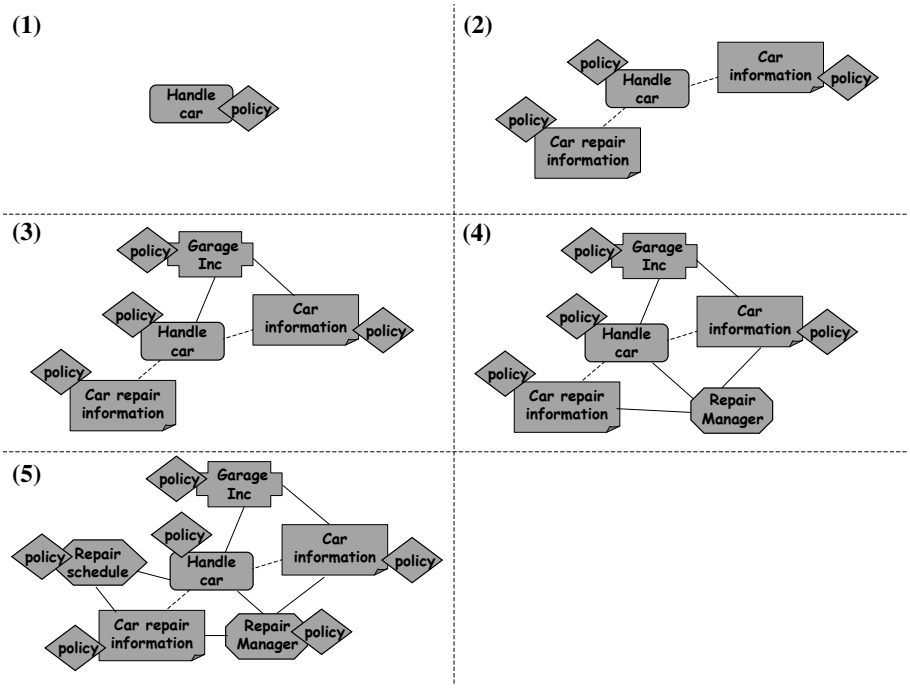

Figure 5.3: A Simple Example Of Rule Based Business Collaboration

defined design based upon which individual business collaborations are created, which are then carried out. Rather, at runtime the appropriate BCIM elements defined in the design schema are specified and combined on an as-needed basis to create the design on the fly, where the combinatory process is driven and constrained by the derivation rules and control rules applicable to the different BCIM elements respectively. This has the advantage that business collaboration designs can be tailored completely to the specific circumstances under which the business collaborations are taking place. This gives organizations like `Garage Inc` and `Lee C.S` the capacity to easily adjust their behavior as the conditions in the business environment change. To illustrate this let us look for a moment at Fig. 5.3.

The figure shows part of the example used in the introduction of Chapter 4 to illustrate the different business collaboration models. Specifically, it shows the development of `Garage Inc`'s internal business process model at the strategic level. In this example we find several BCIM elements like step `handle car`, resource `car repair information` and schedule `repair schedule` in the different boxes (1) to (5). These elements are all part of `Garage Inc`'s schema for the private process of `garage owner`. In Fig. 5.3 each of the BCIM elements is annotated with a policy, which depicts the conditions under which it can be linked and mapped to other elements. The conditions are expressed as rules and are based on the different modeling description atoms in the BCIM, as we will discuss in detail in section 5.4. By utilizing the different types of rule that exist (covered in section 5.3) we can drive and control the design process.

To illustrate, let us start in (1) in Fig. 5.3 where we assume that `garage owner` wishes to perform `handle car` at strategic level. Before we can do this though, we need to ascertain what resources are needed and/or produced, under what time frame this is to be done, at what location, by whom, and so on. Therefore, in (2) we deduce that `handle car` uses resource `car information` and produces `car repair information`. Using these resources' policies we also derive information concerning the resource type and value. Subsequently, in (3) we determine the organization involved (being `Garage Inc`) using `handle car`'s policy, and the selected garage dependance. This location can be influenced e.g. by the home address of the car owner, garage capacity and etceteras. We also determine how it links to the already present resources. This process continues in (4) with the addition of `repair manager` and in (5) with the schedule for `handle car`, i.e. `repair schedule`. The period in this latter schedule may depend for example on the type of car that is to be handled, the owner of the car, and so on. The expansion of the design displayed in the different boxes is constrained by the control rules in the policies of these different BCIM elements to assure that no deadlocks occur, resources are accessible, time frames are feasible, and etceteras.

Although the above example is simple in nature, it shows the essence of the rule based business collaboration approach that we propose: organizations like `Garage Inc` and `Lee C.S` develop design schemas describing their business collaborations with other organizations. In these design schemas BCIM elements are pre-defined in nature and have associated policies containing derivation rules and control rules. The derivation rules depict how each BCIM element is to be defined and combined, wheres the control rules express which definitions and combinations are valid and which are not. At runtime the BCIM elements are then be combined on an as-needed basis driven by their respective derivation rules, thus allowing the design to be tailored specifically to the requirements as mandated by the circumstances of individual business collaborations. At the same time the design process is governed by the control rules of the BCIM elements to ensure consistency of the resulting design. Such approach addresses the requirements we stipulated with regard to dynamicity and consistency as follows:

- Rules are pervasive throughout organizations as we saw already in Fig. 5.2. As such they are prime candidates for handling the diversity of change that can occur in the context of the BCCF. By making these rules explicit organizations can effectively incorporate changes ranging from strategic to service level and from private to agreed upon behavior into their business collaborations in the same manner. Concretely, since each BCIM element has its own policy, the specification of these elements and the manner in which they are connected can be finely tuned by adjusting the rules in the policy. Consequently, the specification of BCIM elements can be easily changed by defining and/or modifying the rules in their policies. This allows organizations to make changes to their different business collaboration behaviors by adjusting the policies of the appropriate BCIM elements in the model schemas underlying these behaviors. As we will see in section 5.3 this requires that the policies of BCIM elements encompass a wide variety of rules.

- When it comes to the flexibility of business collaborations (i.e. the ability to handle changes that are known at design time, and are known to occur at some specific point during runtime), the proposed rule based approach facilitates this through the appropriate definition of derivation rules. Concretely, organizations can define different rules in their design schemas to handle different situations. Then, at runtime the rules will be used to deduce how to behave that are applicable in the specific circumstances. For example, depending on priority `Garage Inc` may decide to send `repair estimate request` to `Lee C.S`'s `claim management service` or to its `priority claim management service` (note that as this is part of the agreement between `Garage Inc` and `Lee C.S` both parties will have to agree on this decision). Another example is the definition of two rules that based on the height of the car repair estimate mandate `garage repairer` to either perform `repair car` or `get approval`. We will discuss support for flexibility in more detail in section 6.2.2 of Chapter 6, where we define the algorithm with which designs can be dynamically generated.

- The second identified form of dynamicity, formal adaptability (relating to the ability to handle changes that are known at design time yet are unpredictable in nature at runtime), is accommodated for in an identical manner as flexibility. To recall, changes in the formal adaptability category pertain mostly to exceptions like a fault message in response to a request. Organizations can define how to handle such exceptions by specifying suitable derivation rules in their design schemas. Then, if this exception occurs, the business collaboration behaves in accordance with the specified rules. To illustrate, for the event that `Garage Inc` sends `repair estimate request` to `Lee C.S` and getting a fault response, `Garage Inc` can define a rule stating that if it it receives such response the request should be sent again. This exemplifies that in the suggested approach it makes no difference whether a change is 'normal' or 'exceptional', as both can be handled in the same manner by the definition of appropriate rules. We will demonstrate this in more detail in section 6.2.2 of Chapter 6, where we show how the algorithm that supports flexibility also caters for formal adaptability.

- The need for dynamism is also met in the proposed approach. Dynamism, as we stated in section 1.1.2 of Chapter 1, deals with the ability to modify existing business collaborations at runtime, and specifically to transform them from the old to the new specification when the underlying design is changed. In relation to the proposed approach design schemas constitute the specifications on which the designs of individual business collaborations are based. Now, because the actual creation of a design in the rule based approach is done at runtime, organizations can influence the definition of the resulting design by changing their defined rules until just prior to their actual application. The possibility remains though that rules are changed after they have have been applied already. In such situations the transformation of existing designs from the old to the new design schema is realized by defining new derivation rules and/or, modifying and/or deleting exiting ones.

Then, through evaluation of the modified rules these changes can be incorporated in the existing designs. As we will see this can include the undoing of already made decisions by removing part(s) of existing designs as well as the specification of rules for compensatory activities.

To demonstrate, let us assume for a moment that `Garage Inc` changes its threshold for when to do `get approval` for a car repair from $500 to $400. Then this is resolved in relation to an existing design by: 1) re-defining the associated rules for when to proceed to repair and when to get approval respectively in the design schema; 2) verifying the existing design whether it deals with a repair in the changed range. If so, then it is possible that `repair car` was initiated whilst approval should have been obtained first. In that case this will be corrected by undoing the 'proceed-to-repair' rule and applying the 'get approval' rule. This has the effect that task `repair car` is removed from the design while task `get approval` is added. In case `Garage Inc` has specified rules concerning compensatory activities for the undoing of `repair car`, then those rules are applied as well. For example, undoing `repair car` may involve performing `update administration` as well as `return car to customer`. The result of the change will be an updated design, which can subsequently functions as the basis again for the carrying out of the business collaboration. We will investigate dynamism in more detail in section 6.3 of Chapter 6 when we introduce the algorithm with which the impact of modified rules on existing designs can be assessed and with which affected existing designs can be updated whilst maintaining their consistency.

- Whereas changes in the dynamism category are expected to occur, those in the undefined adaptability category concern changes that are unknown at design time and occur unpredictably at runtime. Typically this kind of change is not addressed in other works due to the problem that business collaborations are usually too complex to identify and define all possible behaviors at design time (in particular as these may exhibit interactions not foreseeable at design time). This problem is circumvented in the rule based approach as it allows new derivation rules to be defined and applied while individual business collaborations are already running. As such, if an unforeseen situation occurs, organizations can simply add new rules or modify existing rules to handle it. In this manner the design schemas underlying business collaborations can always be extended on an as-needed basis eliminating the necessity to have complete knowledge at design time. An illustrative case is when `Garage Inc` in response to a `repair estimate request` receives an unspecified response message. Then, in order to handle this message `Garage Inc` can define new rules in its design schema. These rules can next be applied to deduce how to handle the response message after which the business collaboration continues. In case the new rules affect already carried out parts of the business collaboration, then this is resolved in the same manner as just described for dynamism. We will return to this matter in section 6.3, where we show that the same algorithm facilitating dynamism also caters for undefined adaptability.

- Organizations will wish to ensure that their business collaboration designs not only meet the stipulated requirements, but also that they are consistent. To recall, consistency represents the need to verify that developed designs are valid, aligned and compatible. In the loose and dynamic design process that we are proposing this becomes even more crucial, for example to avoid situations in which emergent behavior is counter to what is actually feasible. Moreover, as there is no explicit design such situations become much more difficult to detect manually. Also, when the rules of a business collaboration in its design schema change, existing designs may need to be modified potentially compromising their consistency. This is particularly the case when one change leads to another, since it then becomes difficult to manually maintain consistency. Therefore, the proposed approach facilitates automated consistency checking by employing the control rules in the policies of the BCIM elements in a design schema. An example is a situation in which `Garage Inc` specifies/modifies two derivation rules such that, when applied in the same design, they lead to of a deadlock situation between tasks `estimate repair` and `repair car` in its internal business process. If such inconsistency would indeed occur, it will be detected by a control rule for deadlocks present in the policies of these respective tasks. Subsequently, `Garage Inc` can resolve the problem by for example prioritizing the two derivation rules. Another example is `Garage Inc`'s rule that a car repair estimate must never be less than $20. This rule ensures that no such estimate will be made as car repairs are conducted by `Garage Inc`. Consistency checking is covered in detail in section 6.2.1 of Chapter 6.

Based on the above benefits the proposed rule based approach sounds promising as it meets our requirements concerning both dynamicity and consistency. However, for the realization of this approach several issues need to be addressed: 1) what exactly are rules in particular when considered in the context of business collaboration; 2) what types of rule exist and are relevant for the specification of design schemas, i.e. of what types of rule does the policy of a BCIM element have to comprise; 3) how can these policies and their rules be made explicit; 4) how can design schemas be developed such that the resulting designs are accurate and consistent; 5) how and in what order are rules to be applied to generate designs by combining BCIM elements in accordance with their policies as defined in the underlying design schema; and 6) how can changes to the BCIM elements and their policies in a design schema be incorporated into existing designs? To answer these questions the discussion of the rule based approach is divided into two chapters. In the remainder of this chapter we will address the first three questions in section 5.2 to 5.4. The development of design schemas, the application of rules to generate designs for individual business collaborations at runtime and the subsequent management of these designs in light of changes are covered in sections 6.1 to 6.3 of Chapter 6 respectively.

5.2 Defining Rules

Before we can use rules to drive and constrain the manner in which business collaboration designs are developed, we first need to know what rules are. Generally speaking a *rule* is considered to be "an accepted principle or instruction that states the way things are or should be done, and tells you what you are allowed or are not allowed to do" (Cambridge Learner's Dictionary, 2006). As such rules represent a fundamental conceptual construct in the reasoning process of human beings, since they enable us to handle the freedom we have in making choices. Without some sort of rules to guide us, we would not be able to make any decision. Organizations like `AGFIL` and `Lee C.S` are human constructs and as such it comes as no surprise that they rely heavily on their policies and rules to govern the manner in which they conduct their business (like sales policies, processing instructions, and etceteras). The question is though what exactly are these rules?

Loosely speaking, rules are typically thought of and expressed as if-then statements. The if part of a rule is called its *antecedent*, and comprises its *conditions*, that is, what must be true. A rule can have multiple conditions connected by so-called operators like 'and' and 'or'. The then part of a rule is referred to as its *consequent*, and constitutes its *conclusions*, that is, what will be true as a consequence of the fact that the rule's conditions are true. Conclusions of a rule, like conditions, can be multiple in nature. An example of an everyday rule for `Garage Inc` is that "if the estimated repair cost is too high or the customer status is not 'gold', then report the cost to Lee C.S". Conditions are often said to constitute the left hand side (LHS) of a rule, whereas its conclusions make up its right hand side (RHS).

Organizations are human fabrications, and just like rules govern human behavior they also govern organizational behavior. These manifest themselves in business policies, contracts, operating manuals, work procedures, and so on. In the context of business collaboration design we are interested in these latter kind of rules, which we call *business collaboration rules*. In the remainder of this section we discuss these rules. We first discuss the characteristics of rules in general in section 5.2.1. Next, in section 5.2.2 we look at the characteristics of rules in the specific context of business collaboration. Subsequently, we introduce and investigate the notion of rule sets in section 5.2.3, which provide a construct for the definition and grouping of rules.

5.2.1 General Characteristics

Rules can be found throughout business collaborations as we saw already in the introduction of this chapter. In the literature the focus of such rules has been mostly on so-called 'business rules'. Many definitions of these business rules exist. (Ross, 1997) defines a business rule as "a statement that indicates a discrete, operational practice or policy in running a business without reference to any particular implementation technology". Alternatively, the work in (Object Modeling Group, 2006) defines them as "rules that govern the way a business operates, where rules are defined as declarations of policy or conditions that must be satisfied." (Moriarty, 1993) uses yet another definition, stating that it is "a constraint

placed upon the business", whereas in (von Halle and Sandifer, 1991) they are perceived as "natural language sentences that describe data requirements to the business users." More recently (Ross, 2003) propose that business rules are "statements that define or constrain some aspect of the business, which is intended to assert business structure or to control or influence the behavior of the business".

What these definitions have in common is that they have a very limited perception of what a business rule is. They view business rules solely as integrity constraints of some form for the preservation of data. The range of rules we are interested in is much broader than that as we are concerned with all rules that can constrain a business collaboration. These can be data constraints, but may also be e.g. workflow type of rules depicting the order of process activities. Moreover, they also include high level goals and policies, as well as implementation level rules. We refer to this broader set of rules as *business collaboration rules* in the remainder of this dissertation to acknowledge the fact that these rules govern the business collaborations of an organization, and are pervasive throughout the business collaboration context. [1]

Rules can be positive or negative in nature, expressing either that something should or should not be the case. The aforementioned rule of `Garage Inc` concerning when to contact `Lee C.S` about a repair is positive in nature. Its negative variant would be that "if `car repair estimate` is too high or the customer does not have 'gold' status, then do not initiate car repair". Rules are defined in different manners with varying degrees of accuracy, completeness and consistency. Most rules are specified in an informal manner, that is, the terms that are used are not well defined. These are the rules that come up in conversation when trying to come to a decision, and are typically the rules that can be found in the business. Because of this reason such rules are often of an ambiguous nature, where interpretation may differ depending on the context in which a rule is to be applied. Furthermore, they may be inconsistent, imprecise, unreliable, procedural, incomplete and redundant. For example, what is meant by that `car repair estimate` is too high? Too much room is left here for personal interpretation, giving way to the (possible) occurrence of ambiguity.

A more well defined category of rules can be found in laws. Laws constitute vast collections of explicit rules. These rules have been specified in such a manner that they are atomic, declarative and reliable. Firstly, 'atomicity' pertains to the fact that the rule can not be divided in smaller units without losing its meaning. For example, the earlier rule of `Garage Inc` is not atomic as it sketches two circumstances under which the same will be true. We can rewrite this statement into two separate rules, being "If the estimated repair cost is too high, then report the cost to Lee C.S" and "If the customer does not have 'gold' status, then report the cost to Lee C.S". Secondly, a 'declarative' rule is expressed independently of any specific method of evaluation, meaning that it tells you what to do instead of how to do it. Thirdly, a rule is 'reliable' if its claimed relevance can be trusted. This is crucial for business collaborations, as organizations depend on them

[1] From here on we shall use the terms 'business collaboration rules' and 'rules' interchangeable unless explicitly stipulated otherwise

in their day-to-day business activities. It would be costly and inefficient for `Garage Inc` to adopt inappropriate rules for when to contact `Lee C.S` resulting in too many cases in which unnecessary time is spent waiting for approval.

A well known characteristic of laws is though that they remain subject to interpretation. This is because the rules depicted in these laws are ambiguous, that is, their exact meaning has not been decreed. Rather, this is dependent on the situation in which the law is to be applied. To avoid any such ambiguity with rules formalization is used. Formalization entails the arranging of the rules in accordance with a fixed structure. This structure depicts the format and syntax in which rules are to be expressed. Moreover, it usually builds on a formal schema in which the terms in the rules are formally defined as such excluding any possibility of confusion over their meaning. Formalized rules are thus an important step in the right direction compared to the earlier discussed informal and ambiguous rules. In relation to the `Garage Inc` example rule the schema will define what "too high" means, e.g. that `car repair estimate` exceeds $500.

Although formal rules are unambiguous in their interpretation, atomic in nature and defined in a declarative manner, this is not enough to make them suitable to drive and constrain business collaboration. This is because the rules are not executable in nature yet. That is, they are not specified yet in accordance with for example a rule engine, which will administer the rules. Alternatively, it may be the case that they have to be rewritten in order to be encoded into software. These lower level variants of rules are sometimes referred to as technical, programming or automated rules. Note that the recognizability of a rule is often comprised when translating it to a particular implementation. Moreover, the meaning of the rule gets lost. This is one of the main motivations for external rule management, as it allows to keep track of rules independent of a particular implementation.

5.2.2 Advanced Characteristics

Based on the general characteristics of rules presented in the previous section a business collaboration rule can be perceived as a formal statement written in a language that can be understood by business people, which is intended to assert business structure or to control or influence the behavior of business collaborations by stating either what should or should not be the case. It is associated with a precise schema and it is both declarative and atomic in nature. Moreover, it is or can be easily made executable. In addition to these characteristics, business collaboration rules exhibit a number of additional properties. Following the analysis presented in (Grosof et al., 1999) we group these around the notion of 'heterogeneity' and 'expressiveness', where we extend it with the idea of 'modality'.

Heterogeneity deals with the diversity of rule implementations typically adopted by the parties in a business collaboration. There are many widely implemented approaches including OPS5 (Brownston, 1985), JESS (Sandia National Laboratories, 2006), and Prolog (Flach, 1994). This heterogeneity of implementation poses several requirements: firstly, as parties need to be able to exchange their rules in order to come to an agreement, the ability to communicate rules is crucial. 'Communicability' entails the exchange of rules with a shared understanding, which implies that rule definitions are not only declarative

but also 'interoperable'. That is, different people from different parties like `AGFIL`, `Europ Assist` and `Lee C.S` must be able to understand each others rules without having to adopt custom and/or proprietary languages, technologies, and etceteras. The importance hereof is also stressed in (Paschke, 2005).

Secondly, communicatibility and interoperability suggest the need for 'ease of parsing' of the rules that are being communicated. Interoperability and executability dictate integration into the WWW-environment as `AGFIL` and the other parties will rely on public networks (most notably the Internet) for the technical realization of their business collaborations. Thirdly, in addition to heterogeneity concerning implementation the diverse variety of rules that `Lee C.S`, `Europ Assist`, and the other parties abide to in their business collaborations must be taken into account. Related to this last point of heterogeneity is the necessity of expressive power. Expressive power conveys the extent and complexity with which rules can be specified. Expressive power is restricted however by computational tractability, meaning that the rules can be handled by the computerized system. First to be reckoned with is the fact that business collaboration rules tend to be highly dynamic, which implies 'ease of modifiability' and thus 'expressive convenience' for rule specification. The latter prompts the need for usage of 'conceptually natural semantics', i.e. it must be intuitive to developers at the different organizations in the `AGFIL` scenario what rules mean so that they can conveniently specify the rules of their business collaborations.

In addition to conceptually natural semantics the ability to define rules that possess *non-monotonicity* (Antonelli, 2006) is relevant for expressive convenience. Non-monotonic rules (also known as defeasible rules) are common in everyday life representing the kind of inferencing in which reasoners reserve the right to retract conclusions in the light of new information. In business collaborations this can happen all the time, for example to override standard rules with special-case exceptions, to incorporate more recent updates and etceteras (an argument also made in for example (Grosof et al., 1999) and (Antoniou et al., 2004)). It is conceivable for example that `Garage Inc` by default will request approval from `Lee C.S` for repairs above $500, except in situations where the customer status is equal to 'gold'. The second rule then overrides the first rule, which consequently must thus be non-monotonic in nature.

To accommodate overriding behavior *prioritization* is needed, since we require some way of determining which rules may be overridden by other, higher-priority rules. Examples of prioritization include the static rule sequence in Prolog (Flach, 1994) and computed rule agenda in OPS5-like rule systems (Brownston, 1985). The prioritizing of rules is also useful to resolve rule conflicts, i.e. situations in which rules lead to contradicting outcomes. When ill-defined it is possible `Garage Inc`'s rules concerning what to do if `car repair estimate` is greater than $500, results in both `repair car` and `get approval`. In such cases one rule must take precedence over the other rule. Note that monotonic rules are definite in nature and thus can not be overridden. For this reason monotonic rules always take precedence over non-monotonic rules. Only when rules are defeasible, priorities are needed to resolve conflicts. An important side-issue in this regard is the capability to specify the scope of conflict, that is, when something constitutes a conflict between rules and when it does not.

Relating back to ease of modifiability, prioritized conflict handling has a positive effect on what (Grosof et al., 1999) calls 'modularity and locality in revision'. That is, if a rule needs to be updated but only for some specific situations, then this can be done by simply adding a new rule without having to modify the existing rule. To illustrate, for current repairs `Garage Inc` may follow the existing approval rule, whereas for new repairs it introduces a new rule in which the threshold is lowered from $500 to $400. This example implies though that organizations have some way of distinguishing between one variant of a rule and another. It also assumes that organizations can determine which rule is applicable in what situation. Furthermore, it necessitates that organizations can keep track of what change was made, why it was made and when. This suggests the necessity of some sort of revision history.

Another factor to be considered with regard to ease of modifiability is the life cycle of rules. Rules will not always be applicable during their lifetime. Also, at some point rules may become obsolete and thus have to be discarded. For example, the existing approval rule will slowly be phased out by `Garage Inc` as current repairs are completed, which in the end will render the rule obsolete. Situations such as these prompt the need for some form of life cycle management to manage the status of rules. In addition, organizations like `AGFIL` will be interested in the reuse of their rules to ease their modification. The sharing of (often centralized) rules across multiple running business collaborations increases manageability, which in turn brings benefits in terms of cost and effort required for the management of the rules. Such reuse may be limited to the referencing of existing rules, but can also include extension of existing rules. In the latter case well-defined semantics are required to define what it means when one rule extends another rule.

Another relevant issue for ease of modifiability not mentioned in (Grosof et al., 1999) deals with capturing and managing the motivation behind rules. In other words, why does `Garage Inc`' define and apply its approval rule? This question falls into several parts: firstly, what is the reason that the rule exists? Did it for example originate from an organizational policy, was it motivated by legislation or was it prompted by a marketing ploy to address shifting customer demands? Secondly, what is the business gain of the rule? Making a rule explicit and maintaining it comes at a cost and it thus becomes important for `Garage Inc` to find out what the benefits are. These benefits can range from increased flexibility and as such reducing maintenance costs, higher customer satisfaction, and so on. Subsequently, a costs-benefits analysis can help determine whether keeping the rule is beneficial or not. Such information is useful for the people making modifications to the rules.

The latter brings up another point regarding ease of modifiability, being who is responsible for developing and maintaining rules. Two important concepts in this regard are 'ownership' and 'stewardship' (von Halle, 2002). Ownership is pretty self-explanatory and refers to the owner of a rule, which is the entity that was responsible for defining it. `Garage Inc`, or `garage owner` will for example be the owner of the approval rule. However, the person who defined the rule does not necessarily also have to be the one who manages it. In an organization most policies and rules will be (originally) mandated by strategic management, yet it is unlikely that they will concern themselves with the actual

enforcement and management of these policies and rules. Rather, they appoint what (von Halle, 2002) calls rule stewards. Rule stewards are people who have the responsibility of maintaining rules and policies. These persons are thus accountable for the rule management even when they are not the owner. Observe that the rules belonging to an individual organization will typically only have one owner and rule steward, whereas the rules present in the agreements between organizations will have at least two owners and rule stewards (one person from each organization).

A final issue, not explicitly mentioned in (Grosof et al., 1999), is the notion of *modality*. Contributing to a rule's expressive power, a modal is an expression like 'necessarily' or 'possibly' that is used to qualify the truth of a judgement (Garson, 2006). Originally the term 'modal logic' encompassed only the just mentioned expressions. However, it is used more broadly now to cover a wide range of expressions besides modal logic, being deontic, temporal and doxastic logic. Modal logic 'old style' is now typically referred to as alethic logic, and consists of the modals 'It is necessary that' and 'It is possible that'. For example, the approval rule gains additional weight if it states that if `car repair estimate` is greater than $500, then it must be the case that `Lee C.S` is contacted for approval.

Another useful modal logic is deontic logic (Lokhorst, 2006). Deontic logic expresses what is obligatory, permissible and forbidden, notions that are commonly found in agreements between parties in business collaboration. These deontic modalities are conveyed in the statements 'It is obligatory that', 'It is permitted that', and 'It is forbidden that'. The above rule rewritten in a negative style becomes "if `car repair estimate` is greater than $500, then it is prohibited that repair initiated". A third modal logic is temporal logic (Galton, 2006), which denotes over what period a judgement is to be true. This can also be relevant e.g. to convey that a pricing commitment remains valid for a limited period of time. Temporal modalities include 'It has always been the case' (H), 'It was the case' (P), 'It will always be the case' (G), and 'It will be the case' (F). To exemplify, adding a 'F' temporal modality to the approval rule results in "if `car repair estimate` is greater than $500, then it will be the case that `Lee C.S` is contacted for approval".

A fourth relevant form of modal logic is the so-called doxastic logic. Doxastic logic deals with depiction of beliefs and non-beliefs, something that is also potentially useful within business collaborations. For example, `Garage Inc` may wish to make its approval rule less strong by stating that "if it is believed that `estimated repair cost` is greater than $500, then `Lee C.S` is contacted for approval". Note that different logics can also be combined to form more complex modalities. To illustrate, in theory `Garage Inc` could state its approval rule such that "if `car repair estimate` is greater than $500, then it is necessary that it will always be the case that it is prohibited that repair is initiated.

Now, there exist relations between the monoticity and negation of rules, and their modalities. If in a rule its conclusion is necessary, then the rule must be monotonic (since otherwise the conclusion could be overridden at some point which contradicts the necessity modality). The same is true for rules containing obligatory, and temporal 'G' and 'H' modalities. Similarly, rules which dictate that something is possible or permitted will be non-monotonic in nature just like those with temporal modalities 'F' and 'P'. The negative modality 'prohibited' can be expressed as a monotonic rule with a negative conclusion.

The modalities of 'non-belief' and 'belief' can be conveyed by a rule that is non-monotonic in nature, and whose conclusion is respectively negated or not. When a condition has one of the aforementioned modalities, then similar relations apply. Concretely, if a condition is necessary/prohibited, then it must be proven. If it is possible/permitted that a condition is true, proof is required that the negation of the condition is not true. If a condition requires that it is believed, then it must be proven. If non-belief of the condition is required, then its negation must be proven.

Additionally, appropriate usage of non-monoticity allows specification of more advanced temporal modalities such as 'Until something is the case something else is the case' (U). Such modality is expressed by using one non-monotonic rule from which 'b' is derivable, another from which 'a' is derivable taking precedence over the first rule, and where 'a' in the second rule invalidates the condition(s) of the first rule. Then, in a situation in which we have deduced 'b' it will hold until 'a' becomes true. Once 'a' is concluded, then consequently 'b' will be retracted; i.e. 'b' is the case until 'a' is the case. As these relations show modalities can be expressed in terms of monoticity and negation, where the modalities function as convenient labels for the rule stewards responsible for defining and managing the rules. Therefore, in this dissertation we will take the discussed modalities explicitly into account when specifying rules in business terms (detailed in section 5.4.2). However, for their formalization, application and implementation we will rely on the underlying monoticity and negation characteristics (covered in sections 5.4.3 and 5.4.4).

5.2.3 Sets Of Rules

Thus far in this section we introduced the notion of business collaboration rules, and discussed their general and advanced characteristics. Resulting from this discussion is the view of a business collaboration rule as a formal statement written in a language that can be understood by business people, which is intended to assert business structure or to control or influence the behavior of the business processes by stating either what should or should not be the case. It is associated with a precise schema and it is both declarative and atomic in nature. Moreover, it is or can be easily made executable and communicatable, and is easily modifiable. In addition, such a statement may be logically non-monotonic in nature, prioritized and stipulate modalities in its conditions and/or conclusions. Each rule furthermore has several characteristics that help facilitate its management like its status, version, documentation, and etceteras.

Although such a description defines what a business collaboration rule is, the unfortunate thing is that humans do not tend to think in terms of individual rules. As observed in (Chisholm, 2004) it is usually not possible for us to accurately remember the details of more than a few rules at a time. Therefore, we prefer to think about collections of rules that together represent some sort of logic that we wish to apply. Rule sets provide organizations like `Lee C.S` with an intuitive logical structure in which they can organize and keep track of rules. Within business collaboration we find ample evidence of the manifestation of rule sets in organizations, such as the collection of instructions `accountant` is to follow when performing task `select assessor` , the set of conditions under which

`car repair service` offers operation `send estimate`, and so on. We refer to these sets of rules as *policies*. In relation to business collaboration we perceive the purpose of these policies as to describe possible courses of action that an organization may pursue under given circumstances.

A policy itself will typically consist of one or more *policy alternatives*. Such alternatives describe a certain course of action by defining a set of logically related rules such that they govern and constrain (some part of) the business collaboration in a coherent, consistent and meaningful manner. The reason for the usage of alternatives is that it enables organizations to cope with different business scenarios. For example, the policy for `estimate repair` will mandate a different course of action on what to do after its completion depending on the value of `car repair estimate` contained in `car repair report` (being that either `repair car` or `get approval` is performed by `garage repairer`). This illustrates that policy alternatives allow organizations to depict several options for the same situation, where one can be chosen depending on its feasibility in the given situation. In order to prevent confusion concerning what to do alternatives must be mutually exclusive, that is, only one alternative in a policy must be applicable under particular circumstances. This can be achieved by ranking alternatives to indicate preference, where the highest priority alternative is considered to be the default policy alternative to be applied. Another option is that alternatives have one or more guard conditions, which define the circumstances under which they are applicable. For `estimate repair` 's policy alternatives the guards will be based for example on the height of `car repair estimate` in `car repair report`. A combination of preferences and guard conditions is required in case multiple policy alternatives share the same guard conditions.

Policies and their alternatives, like rules, must be understandable by business people, be formal in nature, and be executable, communicatable and easily modifiable. In addition, policy alternatives must assert influence over the business in a coherent and meaningful manner. That is, the rules that an alternative groups should together prescribe a consistent, coherent and clear course of action. Also, alternatives from different policies can conflict with each other, which is the case when one or more rules in these alternatives contradict each other. For example, `Garage Inc`'s repair car and get approval alternatives for `repair car` may be in conflict with `Lee C.S`'s views on this matter. The solution is then to find those alternatives in `Garage Inc`'s and `Lee C.S`'s policies respectively such that their rules are consistent with each other, i.e. do not lead to conflicting results. This implies the need for mechanisms with which inconsistencies in policy alternatives and between alternatives can be detected and resolved. These mechanims will also have to define the scope of conflicts. We will return to these matters in Chapter 6 when we discuss the development and management of policies and rules.

5.3 Classifying Rules

In the previous section we introduced and analyzed the notion of business collaboration rules, and saw how they are typically grouped into policy alternatives in policies to provide

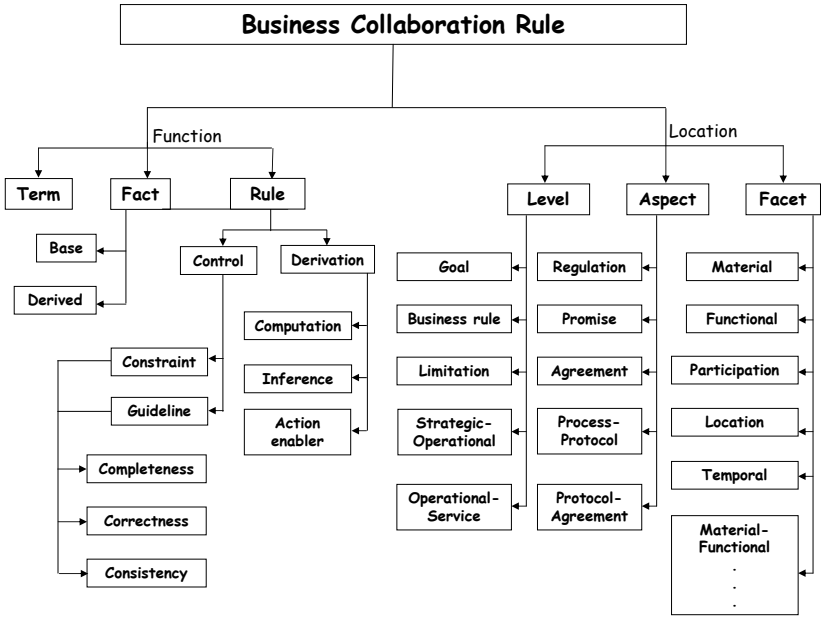

Figure 5.4: Business Collaboration Rule Classification

an intuitive logical structure. During this analysis we casually observed in section 5.2.2 that there exists a variety of business collaboration rules. In this section we present a classification for these rules. The purpose of this endeavor is to help discover and analyze those rules that are of relevance to drive the combinatory process of BCIM elements to develop business collaboration designs. Subsequently we can then make the role and purpose of each type of rule within the policies of the BCIM elements explicit. This in turn gives us the ability to determine how the different types of business collaboration rule exactly influence the development of designs, when to apply what rule and in what order. In short, it tells us how we can employ what types of rule to drive and constrain business collaboration design and execution. It also makes clear which kind of rules must be present in the BCIM element policies for them to be complete.

Having said that, several categorizations have been proposed for rules in literature (specifically business rules), among others in (Date, 2000), (von Halle, 2002), (Ross, 2003) and (Veryard, 2002). Although useful, these categorizations are functional in nature and as such do not provide us with concrete enough insight in what kind of rules are actually of relevance for business collaboration development and management. Therefore we have extended this rule classification by adding a second categorization dimension based on the BCCF. The resulting classification for business collaboration rules is portrayed in Fig. 5.4.

As the figure shows business collaboration rules are divided along function and location. The functional classification is based on the way in which a rule is used, i.e. what its function is. This classification follows to a large degree earlier proposed categorizations in

the literature. We extend the classification though by introducing the categories of derivation rules and control rules. We also divide constraints and guidelines (i.e. control rules) into completeness, correctness and consistency rules to identify the different ways in which these rules can be used to control business collaboration design. Adopting the functional classification also helps to express business collaboration rules in standard rule terminology. This makes it possible to: 1) reason with business collaboration rules using standard rule mechanisms like forward and backward chaining; and 2) to establish a separation of concern between actual business collaboration development and management on the one hand and the application of rules to drive and constrain development and management on the other hand.

The second classification, the grouping along location, helps us pinpoint what part of a business collaboration is driven/governed by a rule. This grouping is done on the basis of the BCCF and is useful in two ways: 1) it helps clarify that during development different types of rule are needed to derive and verify the resulting designs (e.g. who will perform a task versus what operations are used to realize a task). This in turn enables us to assess whether the policies of BCIM elements are complete by verifying what types of rule have not yet been defined; and 2) it assists with the management of these rules as they can be grouped in accordance with their type (for example to get an overview of all task allocation rules). In the following we describe the classification schemes in more detail in sections 5.3.1 and 5.3.2 respectively.

5.3.1 Function Based Classification

In the function based classification business collaboration rules are divided into *terms*, *facts* and *rules*. Terms are defined as nouns or noun phrases with an agreed upon definition, for example a concept such as 'a customer' or a value such as 'female'. In the context of business collaboration design terms define the basic modeling concepts with which models can be created. Terms may be subdivided in literals and types as is done in (Ross, 1997) expressing a specific concept or a type of concept respectively. An example of a literal is a particular `car repair information` resource being exhanged in the `AGFIL-STM`, whereas `car repair information` itself captures the concept of car repair information in general. In this sense literals are thus instances of the modeling description atoms in the BCIM described in Chapter 4.

Facts are statements that connect terms, through prepositions and verb phrases, into sensible, business relevant observations. An example of a fact is 'Female customer can place order'. Facts can optionally be further categorized as well (Ross, 1997), where a distinction is made between base and derived facts, and attributes, generalizations and participations. In this dissertation we only consider base and derived facts, where attributes, generalizations and participations are considered to be special forms of base or derived facts. In the context of business collaboration base and derived facts define how basic concepts relate to each other. For example, the statement that `car repair endpoint` supports `send estimate` is a base fact connecting these terms. A derived fact would be the gross price of `send estimate`, which is calculated by taking its net price and adding taxes. In the

context of the model based approach facts thus reflect how modeling description atoms are connected, e.g. that an element has a certain property or that it is linked to another element.

Rules are then defined through the combination of facts. A rule is viewed as a declarative statement that applies logic or computation to information values resulting either in the discovery of new information or the evaluation of existing information. For example, the rule "if `Garage Inc` is sending a `car repair report` to `Lee C.S`, then this must be done in a confidential manner" enables discovery of new information. In contrast, a rule constraining existing information is that `garage repairer` must not make an estimate of `car repair cost` below $50. In relation to the model based approach in Chapter 4 such rules represent statements that express how modeling description atoms can be combined. Rules are themselves subdivided into *derivation rules* and *control rules*, previously mentioned briefly already in section 5.1. We discuss these two types of rule in more detail in the following.

Derivation Rules

Derivation rules express the peculiarities, originality and values of individual organizations with regard to how they determine in what way to conduct their business collaborations. The explicit definition of such rules enables organizations like `Garage Inc` to analyze their rules, motivate modeling decisions by linking them to corresponding rules, develop different policies for different scenarios, drive development and management using these rules, and if necessary quickly effectuate changes in their requirements for business collaborations by (re-)defining the appropriate rules. Derivation rules encompass a wide range of rules like (among others) data dependencies, control flow statements, quality of service conditions, event/exception handlers, transactional and compensational directives, and so on. Derivation rules can be both stable or dynamic in nature. For example, while we can expect that the rule `get estimate` must be performed by `consultant` will not change over some period of time, the rule concerning the threshold of `car repair estimate` is likely to be subject to change more frequently. As such, derivation rules can be used to capture both stable and dynamic business collaboration requirements in the same fashion.

Following among others (Ross, 2003) and (Wagner, 2002) we divide derivation rules into computations and inferences, and action enablers. Computations and inferences are concerned with deriving new information from some input information. With computations this is done through the application of some algorithm, e.g. to calculate the total sum of the order (product cost plus delivery costs plus taxes). New information is derived through logic in inferences, such as if a customer is of preferred status, then give a 20% discount. Computations and inferences are also often referred to as derivation, deduction or projector rules in the literature (von Halle, 2002). The earlier mentioned calculation to determine the gross price of `send estimate` is an example of a computational rule. An inference rule example is that if `car repair information` constitutes a cost above $1000, then `car repair information` must be send in such a manner that unauthorized disclosure is prevented. Lastly, action enablers initiate new events that fall outside the scope of the

current business event. Reaction rules, event-condition-action rules (ECA rules) and action rules are synonyms for action enablers. In the AGFIL case study `garage repairer` follows an action enabler, which states that if `car repair cost` is below $500, then car repair should immediately commence. In terms of events this means that if `garage repairer` receives an event `estimate repair request` and the result is a cost less than the specified threshhold, then the new event of `start car repair` is initiated. We collectively refer to computations, inferences and action enablers as derivation rules.

Control Rules

Whereas derivation rules enable the derivation of knowledge about business collaborations, control rules function within the rule based approach as guardians of the boundaries of such collaborations. Their purpose is to constrain the development and management of business collaborations in such a manner that these are and remain in accordance with the requirements imposed by the business collaboration domain; and thus that business collaborations are and remain valid, aligned, and compatible. For example, an organization like `Garage Inc` may want to specify that if the customer status is 'gold', then the maximum repair time must be lower than 7 days'. Such rule does not aid with the derivation of knowledge (what the repair period is), but rather constrains the characteristics of such derived knowledge (the possible values for the repair period). By making constrains like these explicit, organizations can use them to ensure that their business collaborations are designed and consequently carried out in a satisfactory manner.

Control rules are either mandatory or optional in nature represented by *constraints* and *guidelines* respectively. This distinction in essence reflects the modalities of alethic logic discussed in section 5.2.2. As such, constraints will be monotonic in nature whereas guidelines are non-monotonic in nature. A constraint that `Garage Inc` might mandate to `garage repairer` is that `car repair cost` can never be less than $100 (in order to cover minimum expenses). In contrast, a guideline can be that if the estimate `car repair cost` exceeds $1000, `car repair report` ought to be send in a confidential manner. These types of rule are often referred to as rejectors or integrity rules in the literature (e.g. (von Halle, 2002)), where they are viewed as guarding the integrity of data. Control rules, like derivation rules can capture both stable and more dynamic business collaboration requirements. To illustrate, we can expect the just mentioned constraint to be dynamic in nature as the minimum repair cost will change over time (e.g. due to increased wages and heightened costs of materials). In contrast, the control rule stating that `car repair cost` must never be below $0 is much less likely to change.

Control rules can be subdivided with regard to their specific purpose into three types of rule: *completeness rules*, *correctness rules* and *consistency rules*. Completeness rules are used to ensure the completeness of business collaboration designs. These rules make sure that: 1) all BCIM modeling description atoms that are needed have been defined in the designs like that `repair car` is allocated to an actor; and 2) that no more than the necessary BCIM modeling description atoms have been defined such as that the access point of operation `send estimate` has been defined twice. The purpose of *correctness rules*

is to maintain the correctness of the definition of individual BCIM modeling description atoms within business collaboration designs. Examples include that step `handle car` is decomposed into tasks and not into events or that the price of operation `send estimate` is always greater than or equal to zero. Lastly, the idea behind the usage of consistency rules is to maintain the correctness of BCIM modeling description atoms in relation to each other, e.g. rules to detect conventional problems like deadlock and looping, improper alignment of security objectives for resource `car repair information` in relation to the associated `car repair report`, and discrepancies of offered quality conditions and agreed upon quality levels of operation `report estimate`. We will return to these matters in section 6.2.1 of Chapter 6, where we will show how we can ensure the validity, alignment and compatibility of business collaboration designs using these three types of control rule.

Returning to the present discussion, as the observant reader will have noticed many of the control rules mentioned above are applicable to all business collaborations. Deadlock for example is a generic phenomenon rather than being specific to particular tasks. As such, many control rules can be considered to be domain independent in nature, that is, applicable independent off the particular business collaboration considered. Such control rules can particularly be found in the meta-models presented in Chapter 4 underlying strategic, operational and service models and the mappings between them. Examples are that steps like `handle car` must be decomposed into at least one task, that the value of a resource such as `car repair information` is in the allowed range of values, that two tasks can not be dependent on each other in order to prevent deadlock, and that a mapping from a task requiring authentication to an operation not supporting an authentication mechanism is not feasible. To enable the sharing of such domain independent control rules across the policies of the different BCIM elements, these rules are pre-defined in the rule based approach. A complete listing of the domain independent control rules that we have identified can be found at (Orriëns, 2007).

The reader is to be aware though that the provided list is not intended to be exhaustive in nature. Also, the list does not imply that all rules are necessary at all times. Control rules pertaining to legal properties may be required for example by `Europ Assist` to verify consistency of legal requirements. Alternatively, perhaps only the set of control rules for checking basic requirements is required by `Europ Assist` in case it does not wish to specify additional advanced requirements for its collaboration with `AGFIL`. Moreover, organizations can include their own control rules in the policies of the BCIM elements in their business collaboration schemas. As such, the set of control rules that an organization uses to constrain business collaboration design, can be both reduced and enlarged in size to customize verification for individual collaborations. Additionally, it is possible for organizations to override the pre-defined domain independent control rules if these rules are contrary to their wishes (or simply not include them in the policies of BCIM elements). For example, whereas a generic control rule might state that the price of an operation must never be less than zero, `Garage Inc` may wish to make an exception for its operation `repair car`. Then, `Garage Inc` can include the control rule in the policies of its other operations, while excluding it from the policy of `repair car`.

5.3.2 Location Based Classification

Rather than considering the function of business collaboration rules, the location based classification in Fig. 5.4 relates to the fact that business collaborations rules can govern different parts of the context in which business collaborations take place. Based on the dimensions of the BCCF rules can then be categorized with regard to the level, aspect and part that they affect. In the following we examine these categorizations in detail.

Along level

Along level rules are divided into strategic, operational and service level rules. These rules guide strategic, operational and service behavior respectively and comprise the policies in the model schemas underlying these different behaviors. Semantically speaking strategic rules capture the strategic requirements of organizations, and are referred to as *goals*. The requirement to prevent unauthorized disclosure of `car repair information` is an example of a goal (in this case security related). Goals make up the policies belonging to strategic model constructs like resources and stake holders, and govern the manner in which these different strategic level BCIM elements can be defined and linked. The meaning of goals is akin to the statements found in high level organizational policies. The main difference lies in the fact that in this dissertation strategic policies are well-defined in nature, while high-level organization policies are usually specified in a vague and textual manner (which makes them more like the goals considered in (Yu, 1997) and (Traverso et al., 2004)).

At operational level in the BCCF, rules constitute *business rules* expressing operational requirements. These business rules in many ways resemble business rules as defined in (von Halle, 2002) and (Ross, 2003), but extend them by not only encompassing data oriented constraints, but also other forms of rule like control flow rules, task allocation decisions, event-condition-action (ECA) rules, and etceteras. An example is that `garage repairer` can work on two cars simultaneously at most or that if event `estimate reported` occurs, then in response `select assessor` must be performed. Groups of business rules constitute policies, which are associated with operational level BCIM elements (such as documents and actors) to regulate the way in which they are specified and linked. From a semantic point of view the rules in such policies represent work procedures, information requirements and so on within organizations.

Service level rules capture *limitations* and express technical conditions imposed on and/or by the IT infrastructure. They are somewhat comparable to for example the assertions we find in WS-Policy (Bajaj et al., 2006), although the constraints here are expressed in a much richter language and can express a greater variety of requirements. Examples of constraints are that `manage claim` must be able to handle at least 100 requests on working days from 9 am to 5 pm, that `repair estimate request` must be sent in an encrypted fashion using a DEA based cypher, and that service `car repair service` must authenticate itself to `claim management service` using a X.509 certificate. Constraints are part of technical policies that define under what conditions services are used, operations are invoked and messages are exchanged. Technical policies thus govern the definition and

linking of service level BCIM elements.

In addition to the above, two other types of rule are identified along level being *strategic-operational rules* and *operational-service rules*. These rules specify the requirements of organizations regarding the mappings between their strategic and operational, and operational and service behaviors respectively. Concretely, strategic-operational rules are part of the policies of both strategic level and operational level BCIM elements, where they govern how such elements may be connected via attributions. Operational-service rules fulfill a similar role, but then in the policies of operational and service level constructs. An example is the rule that event `estimate reported` can only occur if triggers `estimate received` and `estimate acknowledged` have been observed. This is an operational-service rule in the policy of `estimate reported`, guiding the mapping between this event and its triggers from an operational viewpoint. To control the mapping from the service perspective the trigger `estimate received` can have a rule stating that such trigger may only be observed in the context of the event `estimate reported`.

Along aspect

Rules are alternatively classified along aspect, distinguishing them in *regulations*, *promises* and *stipulations*. This categorization is orthogonal to the one along level, and divides business collaboration rules into strategic promises, operational regulations, service stipulations and so on. The purpose of regulations, promises and stipulations is to capture the requirements of organizations for its business processes, business protocols and business agreements respectively, i.e. to express the model schemas for these different models. Concretely, regulations are rules to which an organization internally adheres, and which it does not share with others. Regulations govern the definition and linking of BCIM elements used to model the organization's private processes. Examples are `Garage Inc`'s private strategic regulation to complete `handle car` with high accuracy or `Lee C.S`'s internal guideline about when to request a second opinion of an assessor concerning a car repair estimate.

Promises depict under which conditions an organization is willing and able to cooperate with other parties. Promises are thus the guarantees that one organization is giving to the other. Promises are part of the policies that guide the way in which BCIM elements in protocols are specified and linked. These protocols may be published to let other parties know under what circumstances the organization is willing to collaborate. Semantic wise such promises are akin to the requirements captured in ebXML's collaboration protocol profile (ebXML Initiative, 2002) at operational level, and to those covered in WS-Policy (Bajaj et al., 2006) and Web Service Offerings Language (WSOL) (Tosic et al., 2003) based policies at service level. Examples of promises are "the price `report estimate` will be $50" or "if `car repair service` wishes to invoke `report estimate` the location at which it can be found is the `consultant endpoint`".

Stipulations represent requirements to which organizations have (often contractually) committed themselves. Stipulations thus constitute promises that both parties have agreed upon. They are similar in meaning to for example in WS-Agreement (Andrieux et al.,

2004), Web Service Level Agreement (Ludwig et al., 2003) at service level, and RBSLA (Paschke, 2005) at operational level. To illustrate, a rule found in a service level agreement between `Garage Inc` and `Lee C.S` can state that "if service `car repair service` wishes to invoke operation `report estimate`, then it must provide a X.509 certificate to authenticate itself" as part of the agreed upon policy of the `report estimate` element. Similarly, at strategic level a rule in `car repair information`'s policy may depict that any modification to this resource must be prevented. Stipulations comprise the policies of BCIM elements that together form an agreement between organizations. These are thus the BCIM elements and policies that are shared between `Garage Inc` and `Lee C.S`, and form the overlap between their respective design schemas for the business collaboration.

Lastly, to express the requirements regarding the dependencies among business processes, protocols and agreements *process-protocol rules* and *protocol-agreement rules* are used . Process-protocol rules are part of the policies of the BCIM elements that make up processes and protocols. They govern the manner in which these different BCIM elements can be connected to each other via horizontal attributions. As such, they allow organizations like `Garage Inc` to control how its private processes communicate with the outside and vice versa. An example is a process-protocol rule from event `estimate reported` in `Lee C.S`'s operational protocol stating that if this event occurs, then in its private process event `estimate received` must happen. In a similar fashion are the requirements for the attributions between the BCIM elements in protocols and agreements captured in protocol-agreement rules. These rules enable `Garage Inc` to govern how its protocols relate to the agreements made with `Lee C.S`, e.g. to assess whether these protocols are capable of supporting the made agreements.

Along part

A third form of categorization is based on parts grouping rules into so-called *material rules*, *functional rules*, *participant rules*, *location rules*, and *temporal rules*. This categorization is orthogonal to the other two location based classifications, and can be thought of as subdividing the types identified in these classifications in more specific ones (needed within individual model schemas). For example, we will need to have temporal goals in the model schema for a strategic behavior. Material rules constrain the existence and/or properties of the BCIM elements expressing the material part, i.e. resources, documents and messages. Placed in the context of the works in (von Halle, 2002) and (Ross, 2003) material rules roughly approximate these works' notion of 'business rules' at operational level, where they depict data-like constraints. The rule "the document `car repair report` must be sent in a confidential manner" is an example of a material rule.

Continuing, functional rules govern the properties of step, task and operation BCIM elements, and are thus part of their policies. Functional rules also encompass the well known control flow to structure how things are done, e.g. the rule for `garage repairer` that if `receive information` has been performed, then `estimate repair` has to be done. Participation rules define requirements with regard to who is involved in a process, protocol or agreement. These rules guide the specification of stakeholder, actor and service level

BCIM elements and their properties. To illustrate, `Lee C.S` can define the rule that `claim management service` must support authentication using X.509 certificates. Location rules express conditions concerning the location at which business collaboration behavior is performed, and constrain the properties of organization, unit and endpoint BCIM elements. Lastly, temporal rules impose time-related requirements governing the specification of the properties of schedule, event and trigger BCIM elements (like that event `estimate send` occurs at most 4 days after `Garage Inc` has received the to-be-repaired car).

Further completing classification along part, the rules that govern the relations between parts form separate categories. Concretely, we have *material-functional*, *material-participation*, *material-location*, *material-temporal*, *functional-participation*, *functional-location*, *functional-temporal*, *participation-location*, *participation-temporal* and *location-temporal* rules (Note: not all shown in Fig. 5.4 for reasons of clarity). As each part at a level in BCCF is related to every other part, the policies of the BCIM elements will contain rules in all of the above categories. An illustrative material-functional rule is that operation `report estimate` must have `repair estimate request` as its input, thus affecting the interaction between the material and functional part of `Garage Inc`'s service level protocol.

5.4 Specifying Rules

In the previous section we analyzed a wide variety of business collaboration rules that are required to drive and constrain business collaboration development and management. As a result we now have a clear picture of what kinds of rule we can find and need in the policies of the BCIM elements in the different models in a business collaboration design. In order to be able to utilize these rules though organizations need to have some mechanism to make them explicit. More specifically, in accordance with section 5.2.1 organizations require a language with which they can specify rules as formal statements that are easily readable and understandable, are associated with a precise schema, declarative and atomic in nature, and easily executable. Additionally, conform the advanced characteristics of business collaboration rules identified in section 5.2.2 the language must be capable of expressing prioritizations of rules to facilitate conflict resolvement, handle non-monotonicity, and allow modularity and locality in revision (including life cycle management, versioning, history preservation, stewardship, and so on). It must also be easily parse-able, computational tractable, and have conceptually natural semantics. In addition, the specification of policies as discussed in section 5.2.3 must be supported such that logically related sets of rule can be made explicit, prioritized, and associated with required 'ease of modifiability' characteristics.

After a careful review of current rule language proposals including (but not limited to) formal logics like first order predicate logic and temporal logic, XML-based languages like SWRL (Horrocks et al., 2003) and RuleML (RuleML Initiative, 2006), and programming level specifications such as Prolog (Flach, 1994) and Java Rules API (JSR94, 2006), we were unable to identify a language that is highly readable, has formal semantics, is easily

executable, and meets all the above re-iterated requirements. Therefore we propose a new rule language here in which rule specification is grounded on an unified conceptual model, yet is represented in different manners to accommodate readibility, formalizability, and executability. We have dubbed this language the *Business Collaboration Rule Language (BCRL).*

The BCRL constitutes of three sub-languages: a business language, a formal language and an executable language. Developers express their rules in the business language in terms of semi-natural language. These rules are then translated into formal representations in the formal language facilitating formal rule analysis and verification. Formal rules are themselves subsequently translated into executable statements to facilitate their communication and application. All three sub-languages are based on a generic conceptual model. Thus, every sub-language serves a different purpose yet it is grounded on the same conceptual model. As a result each language will have its own syntax, but it will semantically be capable of conveying the same information.

In the following we first discuss the conceptual model underlying the BCRL in section 5.4.1. After that in sections 5.4.2 through 5.4.4 we describe the syntax of the business, formal and executable language, and illustrate their usage with examples from the AGFIL case study. To illustrate the usage of the three languages and their differences in appearance we will take `Garage Inc`'s policy for its resource `car repair information` as an example. For the sake of illustration we will assume that this policy consists of two policy alternatives. The first alternative defines `Garage Inc`s default requirements for the resource, whereas the second one captures the requirements when dealing with `Lee C.S`. The default policy alternative itself contains two rules: 1) the derivation rule "if the estimated cost of repair is more than $500, send `car repair information` so that it will not be disclosed to anyone besides `Lee C.S`; and 2) the control rule "resource `car repair information` must always have a value of 'true' for its property 'disclosure'".

5.4.1 Conceptual Model

Despite their differences in syntactical appearance and purpose all three sub-languages in the BCRL are grounded on a generic conceptual model. Fig. 5.5 shows an overview of this model.

In the bottom of the figure we find the BCIM modeling description atoms constituting the different business collaboration models, i.e. contexts, elements, properties, links, and attributions as defined in the BCIM in section 4.6 of Chapter 4. In line with what we discussed in section 5.3.1, these atoms function as terms in the BCRL. By making statements about atoms facts can be asserted, to which the BCRL refers as *clauses*. For example, the statement that a property belongs to a certain element constitutes a clause. Using modeling description atoms as the terms based upon which facts can be defined, gives the BCRL conceptually natural semantics as rules can be defined in terms of concepts familiar to developers (being the concepts used in the different business collaboration models). Clauses can have an operator and value in case they constrain the values of properties, e.g. to express that the latency of a task must be lower than one minute. Supported

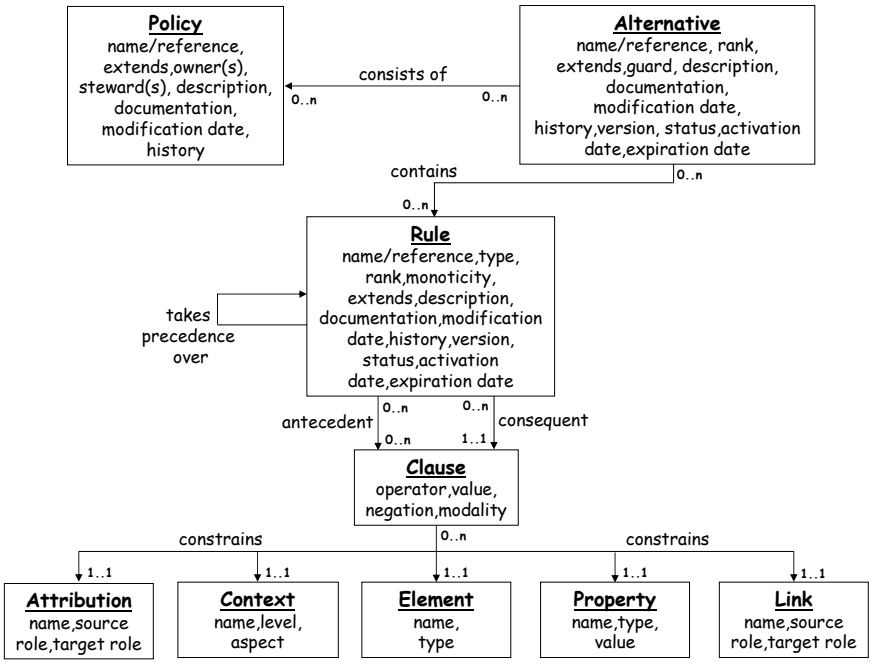

Figure 5.5: BCRL Conceptual Model

operators include numerical comparison operators (like '>' and '=='), text operators (like 'contains', 'equals', 'starts with', 'ends with', 'length', 'uppercase', 'lowercase' and 'substring'), logical operators (i.e. 'true' and 'false'), membership operators (like 'member', 'not member', 'first member', 'last member' and 'sublist'), existence operators (like 'exists' and 'not exists'), math operators (being 'add', 'subtract', 'multiply', 'divide', 'power', 'sinus', 'cosinus', 'tangent', 'round'), and date and time operators (like 'before', 'same', 'after', 'year equal to', 'hour greater than', and etceteras). Derivation rules can only employ operators in their conditions, whereas control rules must use an operator in their conclusion. A clause can furthermore be negated to allow specification of both positive and negative rules. Concretely, rule conditions can be negated in both derivation and control rules. Rule conclusions can only be negated in control rules to express constraints. As such, it is not possible to derive negative facts with rules defined in the BCRL. A clause can also have one or more modalities, where these modalities must be in correspondence with the negation of the clause and the monoticity of the rule in which the clause is contained as discussed in section 5.2.2. Note that modalities are only used in the business language discussed in section 5.4.2.

Clauses can be combined into the antecedent and consequent of rules. Rules are represented using the rule concept and express constraints on how modeling description atoms may be combined. For identification purposes each rule has an unique name. The rule's monoticity conveys whether we are dealing with a logically monotonic or non-monotonic rule. For control rules monoticity conveys the difference between 'hard' constraints and

'soft' guidelines. The conditions and conclusions of rules are expressed in clauses, thus indirectly in terms of the different modeling description atoms. In this manner rules constrain the existence and characteristics of modeling description atoms. The policies of BCIM elements are made up of such rules (the different policy alternatives in these policies to be exact), governing the values of their properties and the manner in which they can be combined with other BCIM elements through links and attributions. Thus, the fact that a rule belongs to the policy of a BCIM element is reflected in the fact that this rule affects the specification of the element somehow (being the properties it has, or the links or attributions it is part of).

Within rules clauses can be connected to form more complex statements. However, to ensure the atomicity of rules conditional clauses can only be conjunctively combined in the BCRL. Also, each rule must have exactly one clause in its conclusion. We are aware that many business collaboration rules will at first be non-atomic in nature containing for example disjunctive conditions or indefinite conclusions. We assume that such rules are rewritten accordingly to express them as atomic statements. A rule with conjunctive conclusions can be split for example in two separate rules, whereas a rule with disjunctive conditions can be split in separate rules as well. This can be thought of as rule refactoring where rules are automatically transformed. How this is exactly done is outside the scope of this dissertation. For more information the reader is referred to for example (Dietrich and Paschke, 2005) and (Paschke, 2005), which identify narrowing, removing disjunctions to obtain clausal normal form, and removing conjunctions from rule consequents via Lloyd Topor transformation (see (Lloyd and Topor, 1984) for the overview of these transformations).

The function of the rule is captured in the 'type' attribute. Derivation rules are of type 'computation', 'action enabler' or 'inference', whereas control rules are of type 'completeness', 'consistency' or 'correctness'. The type of a rule is also reflected in the clauses it uses in its antecedent and consequent. For derivation rules the following applies: computations must employ mathematical operators in their conditions. Action enablers require the constraining of events in their conditions leading to a task in its conclusion. Inference rules can refer to any type of BCIM modeling description atom in its antecedent and consequent. Its condition clauses can define any kind of operator, however, its conclusion can not have any operator (since the conclusion constitutes a derivation rather than an evaluation of a statement). As such, computations and action enablers can be considered to be special cases of inference rules. With regard to control rules the following restrictions exists: a completeness rule can refer to any modeling description atoms in its conditions and conclusion. However, because such rule specifies what BCIM modeling description atom must be present/absent, its conclusion must have the 'exists'/'not exists' operator associated with it. In contrast, the conclusion of a correctness rule can have any operator associated with it with the exception of the existence operators. However, both the conclusion and conditions of a correctness rule must constrain the same element (or its properties, links or attributions). Lastly, the conclusion of a consistency rule has the same operator restriction as a correctness rule, but at least one of its conditions must refer to a BCIM modeling description atom different from the one being constrained (as consistency

rules relate characteristics of different BCIM elements).

Following the discussion in section 5.2.2 the BCRL must also accommodate definition of logically non-monotonic rules as well as modularity and locality in revision. As observed in that discussion non-monoticity leads to the need for prioritization. Prioritization can be done in an absolute or relative manner (Grosof et al., 1999). With absolute prioritization each rule essentially gets a priority label, where on occurrence of a deadlock the labels are compared and the rule with the highest priority is preferred. The static sequence definition in Prolog (Flach, 1994) and priority labels in many OPS5 inspired systems (such as JESS (Sandia National Laboratories, 2006)), are examples of absolute priority scheming. Although absolute prioritization is easiest to implement and maintain, such scheme is not always capable of handling situations where rules with equal priority conflict. Therefore, we choose to adopt both absolute and relative priority specification. Absolute prioritization is conveyed in the 'rank' property of rules (where a higher number indicates a higher rank), whereas relative priorities are defined using 'takesPrecedenceOver' relation between rules. Note that only logically non-monotonic rules can be overridden in the BCRL, be it by other non-monotonic rules or by 'normal', monotonic rules. Monotonic rules are interpreted in the classical sense where conclusions, once drawn, are not retractable. As such, they need not be ranked.

Both derivation rules and control rules can be monotonic or non-monotonic in nature. Monotonic derivation rules are rules whose conclusions, once deduced, can not be retracted. Such retraction is allowed for facts justified using non-monotonic derivation rules. Control rules that are monotonic express constraints that must be true. Thus, if they are violated, these violations must be successfully resolved. In contrast, an attempt will be made to resolve violated non-monotonic control rules, however, this does not necessarily have to succeed. Derivation and control rules also have an associated rank and/or relative priorization statements. For derivation rules they serve the purpose of determining in what order rules are to be applied to derive new information. In relation to control rules ranks and prioritization statements convey in what order detected problems are to be resolved. Concretely, if inconsistencies have been detected using the consistency rules, the violated rules can be ranked to determine in what order to resolve these inconsistencies. With regard to completeness rules, when these are used to verify completeness, the ordering of violated rules allows us to govern the order in which the development of a business collaboration design takes place. As a side note observe that the domain independent control rules we mentioned in section 5.3.1 may also be defeasible to allow them to be overridden by the control rules of individual organizations. We will return to these matters in section 6.2.2 of Chapter 6, where we explain the role of absolute and relative prioritization in the development and management of business collaboration designs in detail.

Continuing here with the discussion of the BCRL conceptual model, rules and their relative prioritization statements are themselves grouped into alternatives, which represent logically related sets of rules. Each alternative has an unique name to refer to it, and contains zero or more rules. An alternative also has a rank, which conveys the priority of an alternative. The reason that we only adopt an absolute prioritization scheme for alternatives is that within a policy the number of alternatives is small. As such, it is quite easy

to rank alternatives without running the risk of having alternatives with the same priority, since we can enforce that each alternative must have a unique priority rank. Moreover, in addition to a rank a policy alternative can also have one or more conjunctive guard conditions that express under what circumstances the alternative is applicable. These conditions may be based on any modeling description atoms excepting the ones describing the BCIM element whose policy the alternative is part of. The reason is that these latter atoms can only be deduced by applying the rules in the policy alternative. However, this can not be done before the guards of the alternative have itself been evaluated, which are based on those atoms. Therefore, to avoid such cyclic reasoning the guards of a policy alternative can not refer to modeling description atoms controlling the BCIM element with which it is associated.

A group of alternatives itself constitutes a policy, where each BCIM element has exactly one such policy. A policy consists of zero or more alternatives. Policy alternatives are mutually exclusive in nature, that is, only one alternative can be applied at a given time. If a policy's alternatives are not ranked, then one of the alternatives will function as default. This will be the policy alternative that has no (or the least restrictive) guard condition(s), i.e. is easiest applicable. A policy can optionally contain no alternatives, i.e. be empty, in which case no rules are applicable yet for the BCIM element with which the policy is associated. With a similar effect a policy can contain one or more alternatives but none of which are applicable as their guard conditions are not met. We will return to these matters in section 6.2.2 of Chapter 6 when we tackle the problem of how to determine which policy alternative to select and apply from a policy within the context of an already existing business collaboration design.

With regard to the ease-of-modifiability related characteristics of rules (as analyzed in section 5.2.2), the BCRL defines several attributes. To express ownership and stewardship of rules corresponding properties are incorporated for the policy concept. This is based on the observation in (von Halle, 2002) to let the allocation of responsibility be based on the logical model underlying the rules. In the proposed rule based approach the rules are based on BCIM modeling description atoms, which are then associated with individual BCIM elements. Therefore, we advocate to assign one or more rule stewards to each BCIM element in a business collaboration design (like a schedule, document or service). Then, any rules defined in the policy of the BCIM element (i.e. not referenced) fall under the responsibility of the assigned rule steward(s). Observe that by defining ownership and stewardship at policy level we assume that all rules within a policy are managed by the same rule steward(s). We believe that this is a reasonable assumption, since these rules form a logically related collection. As such it seems most prudent to place responsibility for their combined maintenance in the hands of one or more individuals.

To provide rule stewards with information about a rule, the BCRL facilitates definition of external documentation references as well as short textual descriptions. The history of a rule functions as a pointer to an overview of the revisions that have been made to a rule. Also the time at which the last change occurred is logged in the 'modification date' attribute. In order to differentiate between different variants of the same rule the BCRL tags each rule with a version number. By convention a higher version number indicates

a more recent variant of a rule in the BCRL. To control the life cycle management of policies the BCRL introduces a 'status' attribute for rules, where status is equal to 'new', 'active', 'dormit' or 'deprecated'. Changes to a rule status are either user initiated or done automatically based on a rule's activation and expiration date. The former governs the transition of a rule status from 'new' to 'active', whereas the latter controls the transition from 'active' to 'dormit'.

Finally, the reuse of rules is accommodated for through referencing or extension via the 'reference' or 'extends' properties of policies, policy alternatives, and rules. Referencing is used to enable 'horizontal' reuse, that is, the reuse of policies, alternatives and rules through a 'pointer-like' inclusion. This means that a policy, policy alternative or rule is either defined in full or referenced in a pointer based on its name. Referencing at policy, policy alternative and rule level will reduce the effort required to define and maintain rules, for example by allowing organizations to define a rule in full once and then reference it where needed. Then, when the rule has to be changed, modification is only required at a single location. This allows rules applicable to a particular BCIM element to be shared across policy alternatives. It also enables reuse through referencing of 'domain independent' control rules across multiple policies and business collaborations (as these control rules are applicable to certain types of elements, e.g. tasks, rather than to specific elements). In contrast, extension is utilized to establish 'vertical' reuse expressing a generalization/specialization relation between policies, alternatives and rules. This enables organizations to create hierarchies of policies, alternatives and rules, which makes their maintenance more easy. In concrete, in vertical reuse one policy, alternative or rule extends another policy, alternative or rule respectively.

The semantic meaning of referencing and extension are as follows in the BCRL: 1) a policy extending another policy inherits all of its policy alternatives and the extended policy's properties (which can be overridden), and can add more alternatives (which can potentially override the inherited alternatives if the added alternatives have the same name). A referred to policy can not be extended in such manner. 2) A policy alternative extending another alternative inherits all the latter's rules and its properties, and can define additional rules (which can potentially override inherited rules if they have the same name). In contrast, a referred to policy alternative can not contain any new rules or override existing properties. 3) A rule extending another rule inherits all of its conditions while adding new ones as well as the extended rule's properties (which may be overridden). In order to ensure that all inherited and/or referred to rules will be applicable in the context of the extending/referring policy the following procedure is follows: each inherited, referred to, or extended rule is updated such that all mentionings of the element to whose policy the rule originally belonged, are replaced by the name of the element whose policy the rule now belongs to (through inheritance, reference, or extension). For example, if the rule for the resource `car repair information` that states that "if the value of car repair cost higher than $500, then ask approval" is inherited by a policy applicable to resource `car repair costs`, then the reference in the rule to `car repair information` is replaced to one to `car repair costs`.

5.4.2 Business Language

In the previous section we outlined the conceptual model underlying the BCRL. Based on this model we defined (as mentioned) three sub-languages, being the business language, formal language and executable language. Concretely, the purpose of the business language is to provide organizations with the means to define rules in an intuitive and appealing manner, yet precise enough to allow their formalization and subsequent execution. Fundamental to the language is the if-then construct. As we observed already in the introduction of section 5.2 humans tend to think of rules in the form "if this is the case, then do this". Adopting such a style for rule definition will increase the readability and understandability of rules, as it results in rule specifications that are conceptually structured in a way natural to developers. For this reason we employ rule representations that are textual in nature and resemble natural language (inspired by among others RuleSpeak (Ross, 2003)). These representations are based on textual description of the formal definitions provided in section 4.6 of Chapter 4 for the different modeling description atoms. This allows *business language rules* to be easily transformed into their formal representations and vice versa.

In the business language the policy of `Garage Inc` for `car repair information` is represented as a textual tuple "Policy$_{carRepairInformationPolicy,carRepairInformation}$: PolicyAlternative$_{carRepairInfoDefault}$ EXCLUSIVE-OR PolicyAlternative$_{carRepairInfoLeeCS}$". These alternatives are themselves captured in more detailed textual tuples. For example, the default alternative for `supply car repair information` is "PolicyAlternative$_{carRepairInfoDefault,1,gc}$", which provides the name, rank and guard conditions 'gc' of the alternative. Also specified in the tuple (but not shown for reasons of clarity) are its version, description, and etceteras. Its contained rules are as mentioned based on textual representations of the different modeling description atoms. To exemplify, let us look at the mentioned derivation rule. This rule is represented as the tuple Rule$_{disclosurerule,defeasible,1,inference}$. In terms of the business language this is phrased as "If the resource carRepairInformation in agfil-stm AND value of carRepairInformation greater than $500 in agfil-stm, Then disclosure of carRepairInformation in agfil-stm is true". The rule is defeasible, that is, it might be overridden by contrary evidence. To exemplify the specification of modalities let us assume for a moment that the conclusion of Rule$_{disclosurerule,defeasible,1,inference}$ has a 'necessity' modal expression associated with it. The textual representation of Rule$_{disclosurerule,defeasible,1,inference}$'s consequent subsequently becomes "Then it is necessary that disclosure of carRepairInformation in agfil-stm is true".

Rule$_{disclosurerule,defeasible,1}$ is an example of a derivation rule, as it allows the conclusion of a fact. Control rules in the BCRL will not have such conclusion. Rather, the stated conclusion will constitute a test of some sort that must be evaluated. For example, to express that we define the rule Rule$_{disclosurecontrol,monotonic,1,correctness}$ as "If the resource carRepairInformation in agfil-stm, Then value of disclosure must be set to 'true'". Observe that it is possible that developers will wish to specify control rules that take the form of "If A, then not B" (note that this will not be the case for derivation rules as their purpose is to derive new, positive knowledge). In the BCRL language such rules can be specified

in this form, as the conceptual model allows for negated rule conclusions in control rules. The question of course is how to provide proper semantics for such rules, something which is difficult due to the fact that in their current format it is not possible to evaluate the truth of these rules through derivation of new facts. We will come back to this issue in detail in section 6.2.1.

As a final remark observe that domain independent control rules are defined in a similar manner as normal control rules. However, because domain independent rules are generic in nature, they do not refer to specific BCIM modeling description atoms. This requires that they are defined in a parameterized manner, i.e. referring to variables rather than literals. To accommodate this we distinguish between names starting with lower case and upper case, where the former are literals and the latter are variables. For example, to express a generic deadlock detection rule we can state "If the task X in M AND the task Y in M AND X dependentOn Y in M, Then Not Y dependentOn X in M". Finally, precedence among rules can be captured using precedence statements like "disclosurerule TAKES PRECEDENCE OVER genericdisclosurerule" (i.e. $\text{Rule}_{disclosurerule}$ is preferred over $\text{Rule}_{genericdisclosurerule}$).

5.4.3 Formal Language

The BCRL business language is useful for rule specification by business people. However, the resulting business language rules lack formal semantics. To remedy this BCRL business language rules can be transformed into the BCRL formal language. As the formal language in the BCRL we adopt a non-monotonic version of standard First Order Logic (FOL). FOL, also referred to as first-order predicate calculus (FOPC) is a system of monotonic logic that adds the idea of quantification to propositional logic. FOL provides formal semantics to business language defined rules, where FOL these *formal language rules* are represented as FOL material implication statements. These statements also consist of conditions and conclusions, which are grounded on the formal definitions of the BCIM modeling description atoms (discussed in section 4.6 in Part 4). The resulting formal rules are then grouped into formal representations of policy alternatives, which themselves form policies.

To exemplify the previous let us revisit the policy of `Garage Inc` for `car repair information`. Formally this policy is defined as "$\text{P}_{carRepairInformationPolicy,carRepairInformation}$: $\text{PA}_{carRepairInfoDefault} \oslash \text{PA}_{carRepairInfoLeeC.S}$". The example $\text{PolicyAlternative}_{carRepairInfoDefault,1,gc}$ for `supply car repair information` in formal terms is then defined as "$\text{PA}_{carRepairInfoDefault,1,gc}$: $\text{R}_{genericdisclosurerule} \wedge \text{R}_{disclosurerule}$." The guard condition 'gc' of the alternative may then for example be defined as "E(carRepairInformation,resource,agfil-stm) $\wedge$ L(MyLink,receives,leeCS,carRepairInformation,agfil-stm)" to express that it is only applicable if `car repair information` is received from `Lee C.S`. Now, $\text{Rule}_{disclosurerule}$ was informally stated as "If the resource carRepairInformation AND value of carRepair-Information greater than $500, Then disclosure of carRepairInformation must be true". The formal counterpart of this rule is $\text{R}_{disclosurerule}$, which is defined as:

$R_{disclosurerule}$: ∀ ResourceValue,Value,ResourceDisclosure
[E(carRepairInformation,resource,agfil-stm) ∧
P(ResourceValue,value,Value,carRepairInformation,agfil-stm) ∧ Value > 500 →
P(ResourceDisclosure,disclosure,true,carRepairInformation, agfil-stm)]

This definition essentially conveys the same information as $Rule_{disclosurerule}$, but here the necessity of the conclusion is only observable from the fact that $R_{disclosurerule}$ is monotonic in nature as indicated by the ←. If a rule is non-monotonic in nature we use ⇐ to indicate that the rule is defeasible in nature (following conventions suggested by among others (Antoniou et al., 2004)). Observe that $R_{disclosurerule}$ is more explicit than $Rule_{disclosurerule}$ by defining the implcitly present quantifications in the latter through definition of the quantification and scope of the variables. In the formal rule variables 'ResourceValue', 'Value', and 'ResourceDisclosure' are universally quantified over the entire rule denoted by ∀. The quantification scope is set by '[' and ']', and ranges the entirety of the rule here.

As a final remark note that in the examples we deviated from the convention that variables are typically denoted by uppercase 'X', 'Y', and 'Z'. Moreover, literals like 'carRepairInformation' are usually defined in terms of lowercase 'a', 'b', 'c' and so on. The reason is that by using more meaningful variable names it is easier to recognize the similarities between the business language and formal language representation of the same rule for the reader. Typically though $R_{disclosurerule}$ will contain only 'x', 'y', and 'z' and 'a', 'b' and 'c' to specify variables and literals respectively, making formal rules like $R_{disclosurerule}$ much more concise than BCRL business language rules such as $Rule_{disclosurerule}$. Lastly, the prioritization construct is captured in the BCRL formal language using predicate 'PR'. For example $Rule_{disclosurerule}$ taking precedence over $Rule_{genericdisclosurerule}$ is formally depicted as PR(disclosurerule,genericdisclosurerule).

5.4.4 Executable Language

The BCRL formal language is suitable for providing formal semantics to informally defined BCRL business language rules. A drawback is that formal rules are not easily exchangeable, something which is desirable since organizations need to communicate and exchange the conditions applicable to their public behavior. In addition formal rules are not directly executable, and can thus not be used to drive and constrain business collaboration design and execution. To overcome these problems we adopt an XML based language as the BCRL executable language, which facilates communication and execution. A complete definition of the BCRL executable language can be found at (Orriëns, 2007), where XML schemas for both the BCIM and the BCRL can be found. Here we will discuss the main characteristics of the language.

Having said that, the policy representation we are proposing in the BCRL executable language to capture policies (such as those of `Garage Inc`) is rather straightforward, and is inspired by other works in this area like WS-Policy (Bajaj et al., 2006), Legal XML (OASIS Legal XML eContracts Technical Committee, 2006), and WSPL (Anderson, 2004) and conform the requirements conveyed already in the BCRL busi-

ness and BCRL formal language. Basically we introduce two types of tags, being <PolicyAlternative> and <Policy> tags. These are defined in the following format (exemplified for $P_{carRepairInformationPolicy}$ of `Garage Inc`, which was defined in section 5.4.3 as $PA_{carRepairInfoDefault} \oslash PA_{carRepairInfoLeeC.S}$):

```
<Policy>
    <Name>carRepairInformationPolicy</Name>
    <Element>carRepairInformation</Element>
    <Extends>genericResourcePolicy</Extends>
    <Owners>
        <Owner>GarageOwner</Owner>
    </Owners>
    <Stewards>
        <Steward>GarageRepairer</Steward>
    </Stewards>
    <Description>Describes how to handle car repair information</Description>
    <Documentation>
        <DocumentationReference></DocumentationReference>
    </Documentation>
    <ModificationDate>01-12-2006</ModificationDate>
    <History></History>
    <PolicyAlternative>
        <Name>carRepairInfoDefault</Name>
        <Guard><Clause>...</Clause></Guard>
        <Rank>0</Rank>
        <Extends></Extends>
        <Description>the default car repair alternative</Description>
        <Documentation>
            <DocumentationReference></DocumentationReference>
        </Documentation>
        <ModificationDate>01-02-2006</ModificationDate>
        <History></History>
        <Version>1.0</Version>
        <Status>active</Status>
        <ActivationDate>01-01-2005</ActivationDate>
        <ModificationDate>01-01-2010</ModificationDate>
        <Rules>
            <Rule><Name>disclosurerule</Name></Rule>
            <Rule><Name>genericDisclosurerule</Name></Rule>
            <Rule>...</Rule>
        </Rules>
        <Overrides>
            <Source>...</Source>
```

```
            <Target>...</Target>
        </Overrides>
    </PolicyAlternative>
    <PolicyAlternative>
        <Name>carRepairInfoLeeCS</Name>
        <Extends>carRepairInfoDefault</Extends>
        <Description>the Lee C.S car repair alternative</Description>
        <Documentation>
            <DocumentationReference></DocumentationReference>
        </Documentation>
        <ModificationDate>01-07-2006</ModificationDate>
        <History></History>
        <Version>1.0</Version>
        <Status>active</Status>
        <ActivationDate>01-01-2006</ActivationDate>
        <ModificationDate>01-01-208</ModificationDate>
        <Rules>
            <Rule>...</Rule>
            <Rule>...</Rule>
        </Rules>
        <Overrides>
            <Source>...</Source>
            <Target>...</Target>
        </Overrides>
    </PolicyAlternative>
</Policy>
```

The above provided XML definition of a policy speaks for itself and we will therefore not discuss it in detail here. However, it is noteworthy to point out that in the above definition all characteristics of a policy and its alternatives are included (in contrast to the earlier provided examples illustrating the business and formal language). Also, it is important to observe that policies can refer to policy alternatives to include them; rather than specifying them from scratch. This is done by replacing the <Name: > tags with <Reference> tags, which allows reuse of rules alternatives across multiple policies. Now, the policy definition is missing the specification of the rules contained within the default alternative of the policy for `car repair information`. To illustrate how such specification is done, when we translate the example rule $R_{disclosurerule}$ into its BCRL executable language representation this results in:

```
<Rule>
    <Name>disclosurerule</Name>
    <Type>inference</Type>
    <Monoticity>defeasible</Monoticity>
```

```
<Antecedent>
    <Clause>
        <ModelingAtom>
            <Element>
                <Name>carRepairInformation</Name>
                <Type>resource</Type>
                <Model>agfil-stm</Model>
            </Element>
        </ModelingAtom>
        <Operator/>
        <Value/>
        <Negation>false</Negation>
        <Modality/>
    </Clause>
    <Clause>
        <ModelingAtom>
            <Property>
                <Name>ResourceValue</Name>
                <Type>value</Type>
                <Value>Value</Value>
                <Element>carRepairInformation</Element>
                <Model>agfil-stm</Model>
            </Property>
        </ModelingAtom>
        <Operator>greaterThan</Operator>
        <Value>500</Value>
        <Negation>false</Negation>
        <Modality/>
    </Clause>
</Antecedent>
<Consequent>
    <Clause>
        <ModelingAtom>
            <Property>
                <Name>ResourceDisclosure</Name>
                <Type>disclosure</Type>
                <Value>true</Value>
                <Element>carRepairInformation</Element>
                <Model>agfil-stm</Model>
            </Property>
        </ModelingAtom>
        <Operator/>
        <Value/>
```

```
            <Negation>false</Negation>
        </Clause>
    </Consequent>
</Rule>
```

Like the XML definition of a policy, the definition of a rule in the BCRL executable language is rather straightforward. Note though that statements such as the one above capture the derivation rules in BCIM elements. Control rules are expressed in the same way, but with the difference that there the clause enclosed in the <Consequent> tags will have an operator and value (conform the discussion in section 5.4.1. Also, policy alternatives can refer to rules to facilitate reuse by using <Reference> tags to identify rules rather than by providing full specifications.

5.5 Discussion

In this chapter we submitted our proposal for a rule based approach for dynamic business collaboration development and management. Concretely, we argued for a style of development in which designs are built from the BCIM modeling description atoms that comprise business collaborations models in a rule based manner. Each BCIM element is associated with a policy consisting of derivation rules and control rules. The derivation rules drive the specification of the BCIM element as well as how it is to be combined with other elements, whereas the control rules stipulate which element definitions and combinations are valid. Collections of BCIM elements and policies form model schemas, where every such schema specifies the building blocks for a business collaboration behavior. Consequently, as a design encompass multiple business collaboration behaviors its corresponding design schema comprises of multiple model schemas. Business collaboration development then becomes a matter for organizations of defining their design schemas by developing the individual model schemas. Then, at runtime designs for individual business collaborations can be generated by defining and combining the defined BCIM elements conform the rules in their associated policies; where using the derivation rules makes the design process dynamic whereas employing the control rules ensures the consistency of the resulting designs.

Subsequently, in the remainder of the chapter we discussed the preliminaries needed for the proposed approach. Concretely, we looked at the definition, classification and specification of the different rules that can be found in business collaboration. When compared to existing works, the described results provide several benefits. To start with, the typical definition of rules is significantly extended in scope. Proposals such as (Moriarty, 1993), (von Halle, 2002), (Ross, 1997) and (Ross, 2003) only consider conventional data constraints, whereas the definition of business collaboration rules in this dissertation is much broader. Other works consider only rules at a certain level to be of interest like goals at strategic level, ECA rules at operational level, or WS-Policy assertions at service level (Bajaj et al., 2006). As such, we address a much greater variety of rules that are of relevance for business collaboration than other related works. In addition, we take advanced requirements such as

negation-as-failure, non-monoticity, prioritization and modalities into account, issues that are not reckoned with in other proposals like Web Service Level Agreement (Ludwig et al., 2003), Web Service Modeling Language (OASIS Semantic Execution Environment Technical Committee, 2006) and Web Service Offerings Language (Tosic et al., 2003). Moreover, we also considered several ease of modifiability related characteristics such as versioning and life cycle management. In contrast, rule based approaches like the ones described in (Casati et al., 2000), (Shankar et al., 2002) and (Zeng et al., 2003) do not consider these issues. This is surprising, since from a practical point of view such issues are of vital important for successful rule management. For this reason these ideas have found widespread adoption in industry, where commercial products like developed by Blaze Advisor (Fair Isaac, 2006), Haley Inc. (Haley Inc, 2006) and ILOG (ILOG, 2006) provide such features in their rule system products.

When it comes to identifying and making explicit the different types of rule that are applicable in business collaboration design, the contribution lies in extension of the default classification of rules as terms, facts and rules (as presented in e.g. (von Halle, 2002) and (Ross, 2003)). Firstly, we explicitly distinguish between derivation rules and control rules to emphasize the difference between their role to drive and constrain business collaboration design respectively. Moreover, we identify completeness, correctness and consistency rules as subtypes of control rules, which (as we will show in Chapter 6) makes it possible to perform specific verification. Although this has also been noted in other works like (Casati et al., 2000), the variety of control rules that these works consider is much smaller. Secondly, we show how rules affect different parts of business collaboration resulting in a variety of rule types each with its own semantics and purpose, where categorizing is based on the dimensions of the BCCF. This provides new insight in the role and function of different kinds of rule in business collaboration.

Another point of contribution is concerned with the definition of a generic language for business collaboration rule specification, the Business Collaboration Rule Language (BCRL). In the literature numerous proposals for rule specification languages can be found. The OGM for example has developed OCL (Object Modeling Group, 2003b) and its proposal for Semantics of Business Vocabulary and Rules (SBVR) language (Object Modeling Group, 2006) (previously known as Business Semantics of Business Rules). SWRL (Horrocks et al., 2003) and RuleML (RuleML Initiative, 2006) are two other contenders, which are both XML-based rule languages grounded on formal logic (where SWRL specializes RuleML to accommodate semantic requirements). These languages can be traced back to IBM Common Rules (IBM, 2002), and are both formal and executable in nature. They also offer support for non-monoticity, prioritized conflict handling and so on. However, they are not very easy to read and understand. They are also lacking support for the definition of characteristics that help increase the ease of modifiability of rules. In contrast, the BCRL business language provides conceptually natural semantics and user friendly text-based representations of policies and rules.

A possible point of critic concerning the BCRL business language may be that the typing in of textual definitions to represent rules, alternatives and policies is a rather cumbersome and potentially error-prone activity. We agree with this viewpoint and for this reason we

envision the usage of a sophisticated graphical user interface in which developers have visual representations of business collaboration models at their disposal (like the ones provided in Chapter 4 for the AGFIL case study), and where they specify policies and rules using intuitive wizards that help them select which graphical elements (representing the BCIM modeling description atoms) are to be constrained and in what manner. We see the role of the discussed business language in this regard as the means with which the policies and rules defined as a result can be presented to developers in a text-based yet user-friendly manner. This will allow developers to easily understand the rules they have specified.

Another category of rule languages mostly focuses on definition of executable statements and thus languages in this category are not very readable. These include the first-order-logic based Prolog (Flach, 1994) as well as JSR-94 (JSR94, 2006) that uses its own proprietary language to provide a JAVA rule engine API. In comparison to these proposals the BCRL is generic in nature, easy to understand, has formal semantics based on the BCIM, and is executable. Moreover, by adoption of an XML based version as the executable language, the BCRL facilitates communication and execution of BCRL based policies and rules on a wide diversity of heterogeneous rule systems. In addition, extension with policy constructs gives us the means to depict semantically highly expressive policies and agreements when compared to other initiatives in this area like WS-Policy (Bajaj et al., 2006) and WS-Agreement (Andrieux et al., 2004), Web Service Level Agreement language (Ludwig et al., 2003) and Web Services Offering Language (Tosic et al., 2003). These works focus on web service level policy specification, and are as such too limited in scope.

A work that comes closer to the BCRL is SweetRules, borne out of IBM's CommonRules project based on the work in (Grosof et al., 1999). SweetRules supports among others prioritized conflict handling and procedural attachments for actions and tests (optionally linked to web service operations). However, SweetRules lacks explicit support for modalities as we use to ease the definition of rules in terms of the business language. Also, to the best of our knowledge SweetRules focuses on operational level rules, and has not been exploited to capture strategic level goals and/or service level constraints. Moreover, SweetRules does not incorporate the notions of policies and policy alternatives. The only thing really lacking from the BCRL at this point, we feel, is its support for interoperability as it is yet another language. However, as of yet no language exists that has the characteristics we identified as being necessary whilst also being interoperable. RuleML (RuleML Initiative, 2006) is envisioned to play this role, but currently does not support all features of the BCRL. If this changes in the future though, then we intend to develop a mapping from BCRL to RuleML to achieve interoperability.

In relation to the questions raised in the fourth of the research questions (as defined in section 1.6 of Chapter 1) this chapter has provided us with the following answers:

1. To begin with, in section 5.2 we analyzed the notion of rules in general and specifically their role, purpose and characteristics in the context of business collaboration. This has given us a thorough understanding of what kind of rules are needed to drive and constrain business collaboration. This has resulted in a clear demarcation of

what business collaboration rules are. We also motivated and explained the usage of policies to group logically related sets of rules. Moreover, we discussed how they are associated with BCIM elements to guide and control the manner in which they are combined to form business collaboration designs.

2. After that we developed a two-tiered classification of business collaboration rules in section 5.3 categorizing rules based on their function and location respectively. The functional classification has helped us to identify how rules can drive and constrain business collaboration design. It has also enabled us to represent business collaboration rules in standard rule terminology. Through the development of the location based classification we have identified the types of rules that are required to drive and control the design of business collaborations.

3. Finally, we devised the Business Collaboration Rule Language (BCRL) to uniformly specify business collaboration rules. To meet the diverse requirements of user friendliness, formal semantics, and executability we introduced a generic conceptual model for the BCRL upon which we based three specification languages: business, formal and executable language. The three languages each serve their own purposes, however, rules expressed in one language can be automatically transformed to another language due to the shared underlying conceptual model.

With the above cornerstones in place we now shift our attention to answering the remaining research questions concerning the application, management and implementation of rule based business collaboration development and management. That is, how can we apply and manage rules to drive and constrain business collaboration development, and how can we implement such rule based business collaboration? We will provide answers to these questions in Chapter 6.

Chapter 6

Rule Based Business Collaboration

Design is a plan for arranging elements in such a way as best to accomplish a particular purpose; Charles Eames

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools; Douglas Adams

In the previous chapter we postulated a rule based approach for business collaboration development and management in which organizations develop so-called design schemas that define the building blocks of designs for business collaborations. These design schemas constitute collections of BCIM elements and their policies. Design schemas are subdivided into multiple so-called model schemas to identify the building blocks for the different models in a business collaboration design. Business collaboration development for an organization then becomes a matter of defining the different model schemas for their private processes and protocols, after which the organization can negotiate with other parties to develop agreements based on those protocols. As such, the designing and running of business collaborations constitutes the activity of combining BCIM elements on an as-needed basis to dynamically create designs at runtime; where the combinatory process is driven and controlled by the derivation and control rules in the policies of these BCIM elements. We next developed an understanding of what the rules in the policies of BCIM elements are in the context of business collaboration and what their role, meaning and characteristics are. We also investigated what different types of rule exist, and examined how they impact the design and execution of collaborations. Subsequently we introduced the BCRL rule language with which organizations can specify the rules applicable to their business collaborations in an informal, formal and executable manner. In relation to the research proposal described in section 1.3 of Chapter 1 this brings us to the fifth step in the research road map, as illustrated in Fig. 6.1

Now, even though the classification of rules and their subsequent specification using the BCRL are all necessary preliminaries we are still a long way of realizing the proposed approach. Concretely, we need the means with which we can generate and manage business collaboration models in order to make their development and management dynamic.

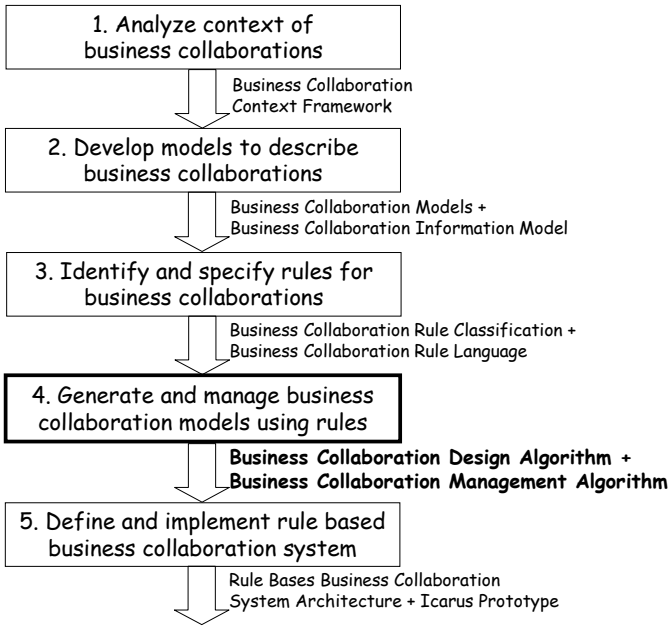

Figure 6.1: Research Road Map - Generating And Managing Business Collaborations

In this chapter we will develop such means through the definition of a *Business Collaboration Design Algorithm* and *Business Collaboration Management Algorithm* respectively. To illustrate why we need these algorithms, let us revisit for a moment the example we introduced in section 5.1 of Chapter 5. To recall, in this example we had several BCIM elements like step `handle car`, resource `car repair information` and schedule `repair schedule`, which were used to describe `Garage Inc`'s internal business process model at the strategic level. More specifically, we stated that by applying the policies of these different elements, they could be combined to form a business collaboration model. Now, in order to intuitively convey the essence of the rule based approach this simple example was sufficient. However, the simplification that we employed to achieve this purpose forced us to omit several problems that need to be addressed in order for the approach to actually work.

A first problem that is of concern is the development of design schemas. In the previous chapter we explained how such schemas comprise of BCIM elements and their policies. We then explored what kind of rules we will require in these policies in order to be able to drive and constrain the design of business collaborations in the manner described in the introduction of Chapter 5. We did not address the issue however of how we can develop design schemas. As business collaborations are complex in nature encompassing three different levels and aspects conform the BCCF, their schemas will comprise of many BCIM elements with potentially highly complex policies. The question is then how organizations can create and manage design schemas in such way that these accurately express their

requirements whilst at the same time lead to complete, consistent and correct business collaboration designs. For example, if `Garage Inc` introduces rules concerning when to perform which task in its operational process, then how can the organization ensure that these rules will not lead to deadlock situations at runtime in the context of an individual design? The same types of issue must be resolved as design schemas change over time. If we let them be handled in a non-automated manner, we will have simply moved the problem from manually making changes to business collaboration models to manually modifying rules and policies in design schemas.

A second problem that must be resolved is concerned with the application of rules conform the specified design schema. That is, with the pre-defined BCIM elements and their policies in place in the design schema, how can we actually utilize these policies to drive and constrain the development and management of individual business collaboration designs? One category of questions that comes to mind deals with the affect of a rule, like how this affect can be observed and interpreted. Specifically, how do we verify that the resulting design is in accordance with the requirements in the design schema ? How do we actually check for example that it is conform requirements that `handle car` produces `car repair information` as output? Another matter is concerned with how conflicts are detected and addressed. What if we would have another step `inspect car` preceding `handle car`, but the latter's policy tells us that `inspect car` is dependent on `handle car` as well. Then we would have an undesirable looping behavior on our hands. Also, what if according to `Garage Inc`'s design schema `car repair report` must be sent in a confidential manner, but in a non-confidential manner according to `Lee C.S`'s design schema? How do we detect this and how do we resolve it? Moreover, how can flexibility and formal adaptability be supported while not compromising the consistency of the resulting designs?

A third problem that requires attention deals with the management of design schemas in relation to existing business collaborations. The rules and policies of organizations will be subject to frequent change. This will not only influence these schemas, but also any existing business collaborations based on them. The question then is how can we assess the impact of modifications to design schemas on existing business collaborations? Put differently, how can dynamism and undefined adaptability be catered for in such manner that the resulting designs remain consistent? For example, if `Garage Inc` changes the rules in the schemas for its business protocols and these affect the schemas underlying the agreements `Garage Inc` has made with `Lee C.S`, then how (if required) can the effects be incorporated into already running business collaborations? These and other questions must be answered in order to achieve the goal of dynamic business collaboration through rule based development and management.

In this chapter we shall deal with the above described issues by providing answers to the raised questions. For this purpose the remainder of the chapter is structured as follows: we start in section 6.1 with a discussion on the development of design schemas, where we define several mechanisms to help organizations to define and manage design schemas that are (and remain) consistent. Next, in section 6.2 we develop the Business Collaboration Design Algorithm for generating business collaboration designs through the application

of the rules within their design schemas. We also demonstrate how flexibility and formal adaptability (i.e. dynamicity during design time) are supported by this algorithm. Then, in section 6.3 we discuss the issues involved with the management of design schemas in particular related to the effects of changes on existing business collaboration designs. Moreover, we develop the Business Collaboration Management Algorithm, which provides support for such management as such facilitating dynamism and undefined adaptability (i.e. dynamicity at runtime). Finally, we contrast the approach with related works and assess the realization of the set out research objectives in section 6.4.

6.1 Developing Business Collaborations

As we explained in the introduction of Chapter 5 we propose an approach in which rules are applied to drive and constrain the development and management of business collaborations. A key assumption underlying such approach is that the rules in design schemas accurately and completely reflect how organizations wish to perform their business collaborations. In order to accomplish this organizations require rule management. Rule management, according to (von Halle, 2002), is focused on leveraging policies and rules, and the need to have these stated, understood, controlled, and dynamically changed by the business stake holders. Thus, what is needed is an environment in which rules can be easily defined and changed. To this end (Ross, 2003) defines rule management as "activities and strategies that aim toward identifying and managing rules in order to among others facilitate rapid change, make business processes more adaptable, and improve communication between business users and IT-professionals". Rule management is beneficial for organizations during the development of new business collaborations (i.e. the definition of their design schemas in terms of BCIM elements and policies) to assess their feasibility. Rule management is also of use for organizations to determine the impact of modifications on the accurateness and consistency of existing design schemas.

Now, the BCRL as discussed in section 5.4 of Chapter 5 already offers support for capturing 'ease of modifiability' related characteristics of policies such as rule status, version, history, and so on. However, what is missing are the means to assess the completeness, consistency and correctness of newly defined design schemas as well as the impact of changes to existing design schemas. Concretely, we lack the mechanisms to analyze the individual policies of BCIM elements as well as how these policies relate to each other. To remedy this situation we build on the extensive work that has been done in identifying the different types of problem that can present itself in rule based systems (like (Aiken et al., 1995), (Bailey et al., 2002), (Baralis et al., 1998), (Leemans et al., 2002) and (Wu, 1993)). Based on these works we follow the grouping made in (Preece et al., 1992) distinguishing between four types of problem (referred to as anomalies): redundancy, ambivalence, circularity and deficiency. In the remainder of this section we examine the anomalies in these categories and discuss how they can be detected. It should be noted that for circularity we focus strictly on the role of negation in cyclic rule behavior. An overview is provided in Fig. 6.2.

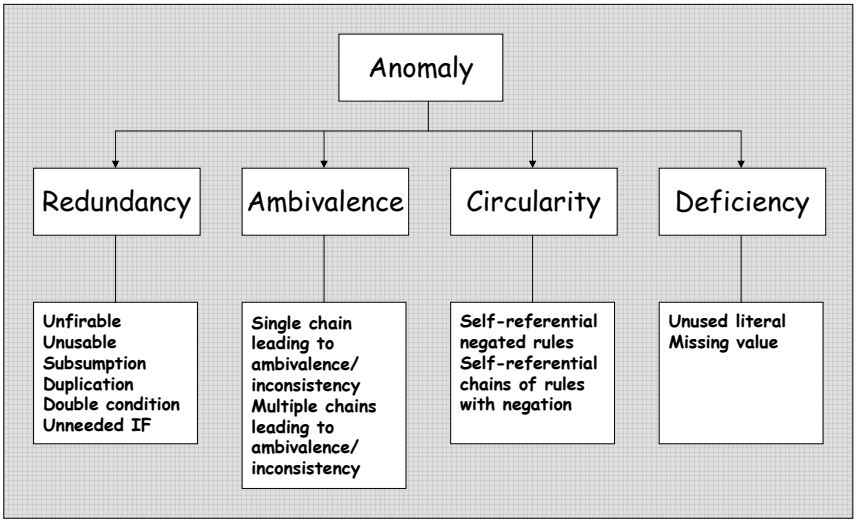

Figure 6.2: Different Types Of Anomaly In Rule Based Systems

6.1.1 Redundancy

Rules (or parts of rules) contain *redundancy* if for every possible interpretation of the rules, i.e. in every possible business collaboration design, it does not matter whether they are applied or not. Detection and removal of redundant rules is important, since it helps reduce the number of rules and the number of parts within rules making policies and their alternatives easier to maintain. Redundancy affects only derivation rules as they are the only rules that enable the deducing of knowledge. Moreover, as all rules affecting a specific BCIM element are part of this element's policy (as discussed in section 5.4.1 of Chapter 5) redundancy can only occur within individual alternatives of a BCIM element policy.

Having said that, the most basic form of redundancy is the so-called un-fireable rule. A rule is un-fireable if it can not be fired, which is the case when its condition(s) are never met. (Bailey et al., 2002) refers to this as the reachability of rules, where a rule is unreachable if no rule(s) exist that make a complete positively contribution to the truth of its conditions. Un-fireable rules are difficult to detect in business collaborations as we can not always say with certainty that a rule will never fire. The exceptions are rules that have conditions that are in violation of the consistency and/or correctness rules. Such rules will never fire, since the required inconsistency or incorrectness will always be prevented. Alternatively, works like (Bailey et al., 2002) consider a rule to be unusable when its consequent is of no use, that is, its conclusion does not appear in any of the other rules. However, within business collaborations it is not necessary that every rule leads to the derivation of information

beyond its own conclusion.

Leaving un-fireable and unusable rules aside for a moment, generally speaking a rule is redundant if it is a logical consequence of other rules. To illustrate, suppose we have the rules "if A then B, and "if B and C, then D". Defining a third rule "if A and B and B, then D" is then not necessary as this indirectly follows already from the two existing rules. The most well known example of this is 'subsumption'. Subsumption of A by B expresses that A and B have the same consequent but B subsumes A's antecedents (i.e. A has more conditions than B). Subsumption becomes a problem if the application of A is what is actually desired, but application of B is achieved (as B will fire sooner than A). Therefore, if a rule B is subsumed by a rule A, and A and B are applicable in the same situation, then B should be removed. A special case of subsumed rules are 'duplications'. Duplicate rules have the same antecedents and consequent, but the order of conditions varies.

Another form of redundancy does not concern rules as an entirety, but rather parts of rules most notably their conditions. A trivial example of this is 'double conditions' where a rule defined as "if A and B and C and A, then D". Obviously inclusion of the second condition 'A' is unnecessary and thus redundant. A more complex instance of redundant usage of conditions can be found when comparing two rules. Referred to by as the 'unnecessary if' anomaly, redundant literals in a pair of rules occurs when two rules have the same consequent, have a conflicting condition, and otherwise have the same antecedents. The conflicting conditions are then unnecessary and in that sense redundant. To exemplify, if we have "if A and B, then C" and "if A and not B, then C", then 'B' and 'not B' can be left out. The result then is a pair of duplicate rules of which one can thus be removed.

All the forms of redundancy discussed so far are syntactical in nature, that is, they occur independent of the actual content of the rules. As (Wu, 1993) and (Leemans et al., 2002) point out semantical redundancies should also be taken into consideration. A semantical redundancy is present if two rules are not redundant syntax wise, but they are in terms of their meaning. In our approach this can only happen if there exist different representations of what is semantically the same concept, e.g. that a rule in one of its conditions refers to the cost of a service and in another to the service's price. One of these conditions is then redundant from a semantical point of view. However, as we stipulated in section 1.5 of Chapter 1 we assume the existence of a shared semantics and as such any redundancies in and among rules can only be of a syntactical nature.

The above leaves us with the task of finding some way to enable detection of syntactical redundancy anomalies. We refer to the mechanism for detecting such redundancies within an individual policy alternative as the 'redundancy detection mechanism'. We adopt the following procedure for this mechanism (Note: we assume that any circularity anomalies have been resolved prior to initiation of the redundancy detection mechanism. Detection of such anomalies is discussed in section 6.1.3):

1. We first check every new or modified derivation rule R within a policy alternative PA whether it is fire-able by verifying that none of its conditions R_{cond} and/or conclusion R_{conc} can be instantiated in such manner that these violate a correctness and/or

consistency rule in PA. This can be determined either by simply testing many possible instantiations or (in some cases) by common sense reasoning. For example, if a correctness rule states that price must be greater than zero and an R_{cond} states that price must be smaller than zero, then R_{cond} can never have a correct instantiation. If indeed R has such R_{cond}, then we mark the rule as being un-fireable; that is, assuming that R and the violated correctness/consistency rule will be active at the same times. If there is only an overlap in the periods in which the derivation and control rule are active, then R will only be unfire-able at certain moments. Also, if the violated control rule is a guideline rather than a constraint, it is up to the rule steward to decide whether this correctness rule should make rule R unfire-able.

2. We next check each remaining rule R in PA for double conditions, i.e. conditions such that R_{cond1}=R_{cond2}. This is the case when R_{cond1} and R_{cond2} constrain a modeling atom 'ma_1' and 'ma_2' such that ma_1=ma_2, with operator 'op_1' and 'op_2' such that op_1=op_2, with value 'v_1' and 'v_2' such that v_1=v_2, and with negation '$n1$' and 'n_2' such that n_1=n_2 (in short whether R_{cond1} is an exact copy of R_{cond2}). We keep track of rules with any such conditions, so that the rule steward can resolve the problem by removing the double condition(s).

3. After that we check each pair of remaining rules R_1 and R_2 for unnecessary IF statements. That is, if $\forall$ $R_{cond1} \in R_1$ it is true that $\exists$ $R_{cond2} \in R_2$ such that R_{cond1}=R_{cond2} (as defined in the previous step of the mechanism) except for one condition R_{cond1} $\in R_1$ for which it is true that R_{cond1}=$\neg R_{cond2}$, then we flag both rules. The $\neg$ here stands for the notion of an inconsistency as detected by the consistency rules in PA (rather than the much more basic notion of logical inconsistency, i.e. 'a' and 'not a'). Also, R_1 and R_2 must be active at the same time. If this is the case, then the rule pair is flagged so that the rule steward can remove the unnecessary IF statements.

4. Subsequently we check for the existence of duplicate rules in PA. Basically each pair of rules R_1 and R_2 is checked to see if $\forall$ $R_{cond1} \in R_1$ it is true that $\exists$ $R_{cond2} \in R_2$ such that R_{cond1}=R_{cond2}; and $R_{conc1} \in R_1$ and $R_{conc2} \in R_2$ are such that R_{conc1}=R_{conc2}. If any are found, then the duplicate rules are marked so that the rule steward can select one and remove it in order to resolve the redundancy. Note that in this step we also clean up any duplicates created by the removal of unnecessary IF conditions.

5. We then check for any subsumption relations between pairs of rules R_1 and R_2. R_2 subsumes R_1 if it is the case that $R_{conc1} \in R_1$ and $R_{conc2} \in R_2$ are such that R_{conc1}=R_{conc2}, $\forall$ $R_{cond1} \in R_1$ it is true that $\exists$ $R_{cond2} \in R_2$ such that R_{cond1}=R_{cond2}, and $\exists$ $R_{cond2} \in R_2$ such that it is not true that $\exists$ $R_{cond1} \in R_1$ for which it is true that R_{conc2}=R_{conc1}. If we find such relation for two rules, we then verify that R_1 and R_2 will be active at the same time. If not, then it is best to keep both rules as they will be applicable in different situations and thus not lead to a redundancy. This may be the case for example when employing different variants of the same rule in the same policy alternative PA. If there is an overlap in the period in which both

rules are active, we then examine their priorities (i.e. their monoticity, rank and prioritization statements). The rule steward is then advised to keep the rule with the highest priority. If no conclusive prioritization is present, then the steward has to decide which rule to keep and which to remove.

6. Finally, we can check for generic redundancies, i.e. those in which one rule is a logical consequence of other rules (as described by (Preece et al., 1992)) through rule extension. Rule extension entails computing every possible firing of rules for a rule set by assuming their truth. Each resulting interpretation is then checked for anomalies. If such anomalies exists it is up to the rule steward to decide whether their occurrence is likely, that is, to assess what the chance is of having a business collaboration in which all made assumptions occur simultaneously thus leading to the redundancy. Rule extension is feasible (Ginsberg, 1988), but computationally expensive (Preece et al., 1992) since all possible execution paths must be examined. Some extensions may even be intractable, i.e. not determinable. In such cases the knowledge and expertise of the rule steward can help to identify possible problem areas by defining and employing heuristics that reduce the number of possibilities to be checked. This was suggested already in for example (Laurent and Ayel, 1989). To this end we employ the information as defined in the design schema to allow rule stewards to simulate interpretations of the rules by generating designs under different circumstances and see if any redundancies occur. An algorithm enabling such simulation is discussed shortly in section 6.2.2.

The above procedure can be used by the rule steward(s) of individual organizations to test the rules governing the processes and protocols of these organizations. Rule stewards from different organizations can also work together using the redundancy detection mechanism to verify the rules governing the collaboration between their respective organizations. In all cases the rule steward(s) can do this by checking the policies of BCIM elements for redundancy within a new or modified policy alternative. This is useful when the derivation rules in such alternative have changed or when the control rules have been modified. Any detected problems with the derivation rules can then be resolved by the rule steward(s) in an automatic manner (un-fireable, double conditions, unnecessary IF and duplicate rules, and subsumption with conclusive prioritization) or manual manner (subsumption with inconclusive prioritization). For example, `Garage Inc` can define a rule to give preference repair of cars whose estimated cost is above $15000. This is not very useful though if such high estimate is never made.

If the correctness rules in a policy alternative have changed, it is possible that derivation rules earlier marked as un-fireable will become fireable and vice versa. In such situations, rather than adjusting the derivation rules rule steward(s) can also decide to modify the correctness rules. Perhaps a correctness rule is such that it is always violated regardless of what derivation rules are verified. Another possibility is that the correctness rule simply was specified wrong resulting in the disapproval of correct derivation rules. To illustrate, suppose `Garage Inc` has changed its correctness rule with regard to `car repair cost`

stating that the height of this cost must always be greater than $500. Such rule will disapprove of many estimates made and as such is a likely candidate to be changed. In short, by utilizing the redundancy detection mechanism when rules are defined and modified, rule steward(s) can ensure the correctness of the derivation rules as well as of the correctness rules.

6.1.2 Ambivalence

Ambivalence represents a category of anomalies that express that we can deduce conclusions from a set of rules that are impermissible. A basic example of this presented in (Preece et al., 1992) concerns the rules "if A then B, and "if A and C, then not B". Then, under the assumption of 'A' and 'C' both 'B' and 'not B' can be concluded thus resulting in an inconsistency, since 'B' and 'not B' are not simultaneously permissible. Inconsistency is considered to be a special case of ambivalence in (Preece et al., 1992), where there is no need to make explicit what is permitted or not because prevention of such logical inconsistency is considered to be a given. Following this four types of anomaly are then identified, being ambivalence following a single chain of inference and different chains of inference, and inconsistency following a single chain or different chains of inference. To exemplify the first type of anomaly, let us suppose we have the rules "if A then B", and "if B then C" and the impermissible set [A,C]. Then, assuming 'A' these two rules lead to a result via a single chain of inferencing that is not permitted. Such result can also be obtained via multiple chains, e.g. if we re-define the second rule to "if D, then C". The other two types of anomaly can be illustrated in the same fashion, only we then obtain a logically inconsistent result like 'B' and 'not B'.

In the context of the rule based approach ambivalence and inconsistency are of use for organizations such as `Garage Inc` to test whether the derivation rules in individual policy alternatives do not contain conflicting rules and thus mandate a clear course of action. Detection of inconsistencies is also of relevance for organizations like `Lee C.S` to check whether alternatives from different policies are consistent, that is, that their respective rules do not contradict. The latter enables `Lee C.S` to for example check the requirements for its business processes, protocols, and agreements for consistency, the alignment of the requirements for its private process defined at different levels, and the compatibility of its protocols with made agreements. All these situations require a mechanism with which inconsistencies both within and between policies of BCIM elements can be identified. Before we discuss such mechanism, let us first relate the notions of ambivalence and inconsistency to those used in this dissertation. What we refer to as inconsistency is called ambivalence in (Preece et al., 1992), where the consistency rules depict what is permissible or not. The notion of inconsistency in (Preece et al., 1992) then relates to what we referred to as logical inconsistency.

As such, in theory the treatment of the four types of anomaly described in the previous paragraph can be done in an identical manner. However, looking ahead for a moment already to section 6.1.3, there is no need to treat logical inconsistency as this is prevented through static rule stratification (as a result of which it is not possible to deduce 'a' based

on '¬a' through reasoning and vice versa). As such, here we only need to address the issue of ambivalence. The mechanism that we use for this purpose is called the 'inconsistency detection mechanism'. The informal idea behind the mechanism is to identify any possibly occurring inconsistencies and present these to the rule steward for revision. Detection of inconsistencies within an individual policy alternative is done as follows:

1. We first take the derivation rules in the policy alternative PA that we wish to test and assume that they are all true, i.e. $\forall$ R $\in$ PA we assume that R_{conc} is true forming the set of possible facts PF. This assumption is motivated on the basis that an alternative is meant to guide the business to follow a cohesive course of action, and thus we may expect its rules to all depict part of this course (an assumption in line with other works like (Preece et al., 1992)).

2. We then apply the consistency rules of relevance for the asserted conclusions (i.e. the consistency rules present in the alternative) to find if no impermissible deductions occur. That is, we check whether on the basis of the obtained definition of a BCIM element (by assuming the truth of the conclusions of all the derivation rules in the policy alternative) any consistency rules become instantiated. Formally we can represent this as: $\forall$ CR $\in$ PA such that its type is equal to 'consistency', if it is the case that $\forall$ CR_{cond} of CR it is true that they are met in PF, and it is not the case that CR_{conc} is true based on PF, then an inconsistency has been detected (Note: we define when a condition or conclusion is met in detail in section 6.2.1).

3. If any inconsistencies are detected, then we identify the derivation rules responsible for the inconsistency; i.e. those rules whose conclusions instantiated the consistency rule. Thus, $\forall$ CR_{cond} we find the R $\in$ PA such that R_{conc}=CR_{cond}, resulting in the set of violating rules VR.

4. Next we check whether the identified derivation rules in VR as well as the violated consistency rule CR will all be active at the same time by examining their status. If this is not the case, then we can be assured that the inconsistency will never occur. More formally, if it is the case that $\exists$ R_1 $\in$ VR with $R_{activation_date1}$ and a $R_2 \in VR$ with $R_{expiration_date2}$, and $R_{activation_date1}$ and $R_{expiration_date2}$ are such that $R_{activation_date1}$ > $R_{expiration_date2}$, then R_1 and R_2 in VR will never be active at the same time (since R_1 will only become active after R_2 has become inactive); and thus no inconsistency will occur.

5. If there exists the possibility that the derivation rules in VR will be active at the same time, then we determine whether they have mutually exclusive antecedents. That is, is it perhaps the case that the circumstances under which one of the rules in VR is fired are exclusive in nature in relation to the other rules in VR? If so, then the inconsistency will never happen. More formally, if there a R_1 $\in$ VR such that there $\exists$ R_{cond1} for which it is true that $\exists$ R_2 with R_{cond2} such that R_{cond1}=¬R_{cond2} (with ¬ expressing an inconsistency as detected by the consistency rules), then R_1

and R_2 in VR have mutually exclusive conditions; and as such no inconsistency will occur.

6. Assuming that the involved derivation rules in VR do not have mutually exclusive antecedents, we then investigate whether the conflict can be resolved with the available prioritization information. That is, the prioritization of the derivation rules is investigated to establish which rule takes precedence. Concretely, first we check the monoticity of each rule R in VR. If all rules R that are monotonic, then no preference can be inferred and we move to the next step (as the conclusions of monotonic rules can not be retracted). In case some rules R are monotonic whereas others are non-monotonic, we then check whether the ranks of these non-monotonic rules in VR are such that there $\exists$ R $\in$ VR whose rank R_{rank} is lower than the rank of all other non-monotonic rules in VR. If so, then it is clear which conclusion will have to be withdrawn. If the ranking of the non-monotonic rules in VR failed to provide a solution, we lastly check whether there are prioritization statements PS in PA such that the rules R in VR with equal rank can be ordered. If insufficient information is present to assess this, then we further investigate the inconsistency.

7. When it is unclear which rule(s) are to take preference in case of a conflict, we examine the definiteness of the inconsistency. If the derivation rules R in VR have exactly the same antecedents (i.e. $\forall$ $R_1 \in$ VR it is the case that for each of its R_{cond1} it is true that $\forall$ $R_2 \in$ VR such that R_2 is not R_1 it is the case that it has a R_{cond2} for which it is true that R_{cond1}=R_{cond2}), the inconsistency will be unavoidable and the rule steward must make adjustments (or change the consistency rules). Otherwise, if the rules have different antecedents (i.e. the conflict is only potential in nature), it can not automatically be determined that the inconsistency will occur. Rather, it is left to the judgement of the rule steward(s) to assess the likelihood of such event. Also, if the violated consistency rule CR is a guideline rather than a constraint, then it is left to the judgement of the rule steward whether the inconsistency will need to be resolved or not. Therefore, in both cases we put the inconsistency on the to-be-resolved list.

When all inconsistencies have been analyzed, we present the list of to-be-resolved inconsistencies to the rule steward(s). The rule steward(s) can then take appropriate action, for example by re-defining derivation rules, adding prioritization statements, and so on. This thus allows rule stewards to verify the consistency of newly defined derivation rules as well as of modified ones. For example, let us assume that `Garage Inc` defines a rule stating that if `car repair cost` is equal to $500, then `car repair report` must be sent to `Lee C.S` in a confidential manner. `Garage Inc` also stipulates that `car repair report` will be communicated as a payload of `repair estimate request`. However, if `Garage Inc` already has a rule mandating that `repair estimate request` will be sent without encryption, then an inconsistency is detected by the informally specified consistency rule "if a document is sent in a confidential manner, then the message transporting it must use some form of encryption".

It is also possible that new consistency rules have been added or existing consistency rules have been changed. The inconsistency detection mechanism allows to assess the effects of these changes on the derivation rules. It is possible for example that due to changes by `Garage Inc` to the consistency rules new inconsistencies are detected or existing ones have disappeared. It is also conceivable that due to modification to some consistency rules there do not exist derivation rules at all such that these consistency rules are not broken. By informing the rule steward(s) about what consistency rules underly the inconsistency, `Garage Inc` can easily determine whether this is the case or not. If so, then the rule steward(s) can adjust the consistency rules. Observe that in case these adjustments relate to those parts of the design schema of `Garage Inc` that comprise its agreement with `Lee C.S`, then rule stewards from both organizations will be involved in the modification process.

The described procedure for inconsistency detection is of use to the rule steward(s) for the verification of individual policy alternatives. However, often the conditions of the consistency rules within a policy alternative will not only refer to modeling description atoms belonging to a single BCIM element. Rather, typically they will express constraints on the relation between characteristics of multiple BCIM elements. In order to verify such consistency rules within a policy alternative we need to be able to compare policies of multiple BCIM elements to see if their contained derivation rules are consistent. We can facilitate this by extending the above introduced inconsistency detection mechanism. In the context of the development of design schemas for business collaborations such extended detection mechanism helps to assess whether the policy alternative of one BCIM element is consistent with that of another BCIM element, and consequently whether there exist two consistent alternatives at all. This allows inconsistencies between policy alternatives to be detected within the design schema of a business collaboration before the contained derivation and control rules are actually applied. As such, the chances of encountering an inconsistency during business collaboration design can be minimized by detecting and remedying the problems in advance.

For example, for the internal business process description of `garage repairer`'s behavior a rule steward can analyze if `car repair report` may be received by this actor without this causing some inconsistency. Rule stewards can also test the consistency between BCIM elements from two model schemas describing the same aspect at different levels, e.g. whether the operational and service level policy alternative of the to-be-mapped task `report estimate` and operation `send estimate` in `garage repairer`'s and `car repair service`'s protocol are not conflicting. Moreover, between BCIM elements in model schemas for different aspects rule stewards can examine if the policies of these elements are consistent. To illustrate, `Garage Inc` can assess whether the conditions applicable to the offered operation `send estimate` satisfy the requirements stipulated in its service agreement with `Lee C.S`. The latter is useful in the negotiation process between the rule stewards of `Garage Inc` and `Lee C.S`, e.g. when they are developing a new agreement based on their respective protocols. It also helps rule stewards to assess the impact of changes to an existing agreement on the protocol of each organization (and vice versa). To exemplify, if `Garage Inc` and `Lee C.S` agree on the new requirement that `repair estimate request` must be sent in an encrypted manner, then both parties can verify the

impact on their respective protocol schemas; and if necessary adjust these schemas as well as the schemas of their private processes accordingly.

The actual testing of two policies for inconsistencies is carried out in a cross product fashion. That is, each alternative from a policy is compared with every alternative from the other policy (similar to the procedure described in (Nolan, 2004)). Note that the more choices that exist in the two policies, the more potential combinations could arise. The procedure is as follows for the 'extended inconsistency detection mechanism':

1. We take an alternative PA from each policy P and merge their rules R_1 to R_n. In this merging process we remove any redundancies to clean up the resulting set of rules following the procedure discussed in section 6.1.1.

2. We then test the resulting set of rules RS for inconsistencies following the above described procedure. If any definite inconsistencies exist, then the alternatives are not compatible. Definite inconsistencies are those that can not be avoided, which is the case when contradicting rules have the same conditions and are active at the same time. Potential inconsistencies are left for the judgement of the steward(s) responsible for the respective policies. The exact comparisons made are as described previously in the basic inconsistency detection mechanism.

3. We repeat (1) and (2) for each pair of alternatives PA_1 and PA_2. Once we have evaluated all possible combinations, we determine if there are any consistent alternatives. If not, then the policies P_1 and P_2 are inconsistent.

4. Otherwise, we evaluate the available combinations PA_1-PA_2 to determine if they are applicable at the same time. We do this by examining their guard conditions GC_1 and GC_2. If these are such that GC_1=GC_2 (or one set of guard conditions subsumes the other), the alternatives PA_1 and PA_2 are consistent. If the respective guards are mutually exclusive, then both alternatives can not be applied at the same time and the combination must thus be ruled out. In all other situations it is left to the responsible rule steward(s) to determine whether the two alternatives PA_1 and PA_2 will be applied at the same time or not.

Note that the insight provided by the above procedure is limited by the fact that many of the derivation rules will often depend on many pieces of information, making it hard to automatically assess whether they will actually be fired or not at some point. The only way to address this is through the usage of rule extension checks (mentioned in the previous section), which entails that every possible inference chain in a set of rules is computed and then checked for consistency. If any inconsistencies are found, the rule steward(s) must then decide whether the made rule extension is in fact likely to occur or not. Here as well the rule steward(s) can be helpful by providing heuristics that can help reduce the amount of computation necessary.

Like we did for redundancy checking we tackle the rule extension problem via simulation. Specifically, the rule steward(s) of for example `Garage Inc` can test the new and/or

modified rules by simulating the design of a business collaboration using the new/modified design schema. Any problems will then be detected as the business collaboration is being simulated after which these can subsequently be remedied. For example, `Lee C.S` can simulate how its operational `claim management process` will map to the protocols it exposes to `Garage Inc` and `AGFIL` and verify that these protocols are supported by this process (i.e. there are no inconsistencies between the protocols and the process). `Lee C.S` can also test the consistency of each protocol with the corresponding agreement, made with `Garage Inc` and `AGFIL` respectively. If no inconsistencies are found there as well, then `Lee C.S` can be assured that under the simulated conditions its interactions with both parties will not result in problems.

In contrast, suppose that `Lee C.S` finds that the agreement with `Garage Inc` is not consistent with its protocol in the simulation. In such event the rule stewards from both organizations will have to come together to discuss the to-be-made changes. The inconsistency detection mechanism then will help `Garage Inc` and `Lee C.S` to identify and resolve inconsistencies (e.g. through prioritization) between their respective design schemas. The result will be two design schemas that are consistent by themselves and in relation to each other. As such, during actual design at runtime no irresolvable inconsistencies will be encountered by both organizations, since conclusive prioritization for any occurring conflicts will have been defined. Note that `Lee C.S` can also simulate the effects of the changes made to the agreement schema on the consistency with its protocol schema. If so, then `Lee C.S` can verify in the same manner whether its private process is still consistent the modified protocol. If the private process has to be adjusted, then `Lee C.S` can assess the impact of these adjustments on the protocol schema it exposed to `AGFIL`; and subsequently if needed the consequences of changes to this protocol schema on the agreement schema that is shared with `AGFIL`. In this manner `Lee C.S` can test the consistency of its business collaboration with different parties as the applicable rules change. The question as to how to simulate the design process is answered shortly in section 6.2.2, where we introduce an algorithm for the generating of designs based on their design schema.

6.1.3 Circularity

A third category of anomalies is centered around the notion of *circularity*, which focuses on problems relating to the occurrence of looping of rules as they are applied. Recognition and formal definition of generic circularity is described in among others (Leemans et al., 2002) and (Preece et al., 1992) . There are several specialized forms of generic circularity like described in (Baralis et al., 1998), (Leemans et al., 2002) and (Preece et al., 1992). The most basic ones are *self-referential rules*. These are rules whose consequents are part of their antecedents, e.g. "if A then A". Looping then occurs if there is some way to deduce the first 'A' without having to use the self-referential rule. After that the rule may fire indefinitely. Slightly more complex are 'self-referential rule chains' in which there are two or more rules that together constitute a loop. An example is "if C and D then A" and "if A and B then C, where assuming 'B' and 'D' may result in a non-terminating cycle. It is not difficult to see that as rule chains increase in length, it requires more and more

effort to identify circular references.

Now, it must be observed that circularities as the ones mentioned are not so much an inherent problem of the BCRL rule language as it is of the manner in which the rules specified with this language are applied. For example, the self-referential rule "if A then A" in theory could cause a non-terminating cycle. However, this can be easily prevented by having the rule engine not attempting to proof 'A' again, since it has already proven 'A'. A similar sort of solution can be used when the rule engine encounters cycles ranging over multiple rules. An important issue that can not be so easily addressed though is when negation is involved. For example, suppose we have three derivation rules "if ¬A then B", "If B then C" and "If C then A", and an empty list of facts. Then by lack of proof for 'A' we can deduce 'B'. 'B' however leads to 'C' and then to 'A'. Thus the result is that we end up deducing 'A' on the basis of '¬A'. This is obviously not desirable. The problem here is that these rules are not stratified. Rules are stratified through a process of stratification, where the goal is to avoid situations in which it is possible to start with '¬a' and derive 'a' following the rules. This type of behavior can occur within individual policies of BCIM elements in a design schema, but also emerge as a result of interaction between the policies of different BCIM elements in a model or design schema. Note that in theory the reverse must also be avoided, that is, deriving '¬a' based on 'a'. However, since derivation rules can not have negated conclusions such situations will never occur with BCRL defined rules.

To detect this type of circularity anomaly we utilize the 'circularity detection mechanism', which is aimed at identifying and analyzing self-referential rules and rule chains with negation operators within a given set of rules. The procedure is as follows (and is conform the process of stratifying a rule set):

1. We first check each derivation rule R in the set of rules RS for negated self-reference by examining its antecedent R_{cond1} to R_{condn} and its consequent R_{conc}. If it is the case that there $\exists$ R_{cond} such that it constrains the modeling atom 'MA' with the operator ' not exists' (i.e. '¬MA') and R_{conc} constitutes 'MA', then R is an invalid self-referential rule. Any detected anomalies are presented to the rule steward, so they may be resolved.

2. Then, we attempt find any chain of derivation rules that constitute a circularity with negation anomaly. This reasoning takes place by creating a dependency graph (also known as triggering graphs described e.g. (Baralis et al., 1998) and (Liu, 2001)) of the set of rules RS as follows: first we take the different relation constants used in the rules, that is, the predicates (and their terms) occurring in the rules (i.e. the different modeling description atom definitions referred to by the rules). Then, we draw an edge from relation constant rc_2 to rc_1 if there is a rule R with rc_1 in the consequent and rc_2 in the antecedent. We label the edge with 'negated' if rc_2 is a negated statement. After that we inspect the resulting dependency graph for any cycles involving a negative edge. The rule set RS is stratified if no such cycles exist. Alternatively, if cycles do exist, we examine whether all the rules in a cycle will be active at the same time (in accordance with their activation and expiration dates as

well as those of the enclosing policy alternatives). If this is not the case for all cycles, we also consider RS to be stratified. In the event that there remain resulting cycles, these are then presented to the rule steward for revision.

The circularity with negation detection mechanism is useful to find and correct circularity with negation anomalies both within and between new/modified policy alternatives . Within an alternative the chance of the rule steward(s) accidentally defining a set of rules containing such anomaly is relatively small, although this can surely happen when new rules are added and/or existing rules are modified. However, when taking policies for multiple BCIM elements (possibly maintained by different rule stewards from different organizations) into consideration the likelihood of circularity anomalies increases. To illustrate, suppose that `Garage Inc` has defined the following rules: 1) if `estimate repair` is not performed, then do `repair car`; and 2) if `repair car` is performed, then do `assess damage`. These rules belong to the policy of `repair car` and `make report`. Now suppose that there is a rule in the policy of `estimate repair` stating that if `assess damage` is performed, then do `estimate repair`. Now, at some point it must be derived what task is to be carried out. According to the first rule this is `repair car`, assuming that `estimate repair` has not been performed. Once this task has been completed, the second rule dictates that `assess damage` is conducted. However, upon completion the policy of this task mandates that `estimate repair` is to be performed.

Without stratification of the rules as described above the business collaboration will then end up in the undesirable situation that reasoning based on not making an estimate of the repair costs task leads to the conclusion that such estimate has to be made! However, when the rules are stratified, there will always be an unique interpretation. Observe that an alternative for the described static stratification is dynamic stratification, which achieves the same but then by consistently assigning numbers to predicate symbols to guarantee that a minimal model exists. In such assignment two procedures are followed: 1) if a predicate P is positively derived from a predicate Q, then the stratification number of P must be greater than or equal to the stratification number of Q; and 2) if a predicate P is derived from a negated predicate Q, then the stratification number of P must be greater than the stratification number of Q. This has the effect that when the minimal model is constructed, its core is always preserved and can not be altered (due to the fact that assumed negation-as-failures become invalid and thus any conclusions based upon these assumptions would have to be retracted).

Besides the prevention of such situations the dependency graphs resulting from stratification can also be utilized for a second purpose; being to visualize the cohesion of a set of derivation rules. With cohesion here we mean the degree in which rules have a positive contribution to each others application. This becomes apparent in the number of directed edges among rules in the graph. The higher the number of such edges the higher degree of cohesion potentially will be. Such indicator is useful for organizations like `Lee C.S` to analyze individual policy alternatives as well as of combinations thereof. As an alternative in the policy of a BCIM element like `consume repair information` mandates a course of action, it seems reasonable to expect that its derivation rules will exhibit a high degree of

cohesion. The cohesion indicator can also be of assistance to judge cohesion among policy alternatives in a design schema by relating the derivation rules of these alternatives in the same triggering graph. This is useful for example for `Garage Inc` to assess whether the different BCIM elements in the model schema underlying `garage repairer`'s private process make a good combination or not on the basis of the cohesion among their respective policies.

In addition to the need for properly stratified rules a second type of circularity anomaly that is to be avoided involves the definition of rules utilizing operators with which new facts can be deduced. For example, if we define the rule "if P(n) AND add(n,1,m), then P(m)", then when we assume 'P(0)' this will result in an infinite reasoning; as the set of numbers is in principle infinite. The only operators in the BCRL that allow derivation of new facts are the set of numerical operators, since all the other operators only cause the evaluation of existing facts rather than the derivation of new facts. To identify circularity anomalies involving numerical operators we essentially repeat the steps outlined for stratifying a rule set. That is, we construct a dependency graph for all relation constants, where we annotate edges between constants if they employ a numerical operator. Then, it is not allowed to have any cycles involving a 'numerical operator' edge. Found cycles are checked first to see if all involved rules are active at the same time. If so, then such cycles are presented to the rule steward for revision. Identification of these anomalies is also part of the aforementioned 'circularity detection mechanism'.

6.1.4 Deficiency

The fourth category of anomalies is based on the notion of *deficiency*. A set of rules is deficient if there is a permissible interpretation such that it contains a fact that can not be deduced from the rules. As (Preece et al., 1992) points out this can happen in two circumstances: 1) because of occurrence of a loop, and 2) because of missing knowledge. The first reason is ruled out through application of the circularity detection mechanism as described in the previous section. The identification of the second reason, that of missing knowledge, requires that one must know prior to rule application what the final information derived from the rules must be. In the context of business collaboration this means that for each business collaboration we must establish what information must be contained in its design. Concretely, every BCIM element in a design schema must have a policy such that each of its alternatives allows the derivation all information that has to be defined about this element .

(Preece et al., 1992) observes that such notion of deficiency is difficult to use in practice, because mostly the set of rules is the only known specification. The suggested work-around is to assume for each permissible interpretation of a rule set that some final facts must be deduced. Under that assumption two symptoms of deficiency are the presence of 'unused literals' and 'missing values' (as well as the earlier described unfireable and unusable rules). Unused literals are facts that can be deduced from the consequents of rules in a rule set, but are not themselves part of any antecedents of other rules in the set. Missing values point to absence of rules for facts that have limited variation. For example, if a parameter may have

value 'true' or 'false' but there is only a rule with which 'true' can be derived, then a rule for 'false' is missing. We opt for a different approach here that is not generically applicable to any set of rules, but is sufficient in the context of this dissertation. The approach is grounded on the usage of the completeness rules in the alternatives of the policies of BCIM elements. To recall, the completeness rules depict the information related to the BCIM element that must be present in a business collaboration design. As such, we can use these rules to determine whether the information that can be deduced from the derivation rules within a policy alternative is complete or not.

Based on this notion we define the so-called 'deficiency detection mechanism' to find deficiency anomalies within a policy alternative. This mechanism follows the procedure below:

1. We first assert the consequents of each derivation rule R defined in the policy alternative PA, resulting in the set of possible facts PF. Note that we assume that any redundant rules have been removed already from the alternative, and that these rules have been statically stratified.

2. Next we check PF using the relevant completeness rules. Any facts that should be present according to these rules but which can not be found, are noted and presented to the rule steward. Concretely, for any completeness rule CR $\in$ PA if it is the case that all CR_{cond} are met in PF, and CR_{conc} is also met in PF, then the completeness rule is met. If there exists a completeness rule CR in PA such that it is not met, this means that an incompleteness has been detected. The rule steward must then introduce new rule(s) to address this incompleteness.

3. If required, then all facts in PF that were found to be derivable can be checked via rule extension, where the goal is to assess whether their derivation is certain or probabilistic in nature. This is essentially done by simulating the generating of designs like described in sections 6.1.1 through 6.1.3. If such simulation turns out to be computationally too expensive (or in fact intractable), then rule stewards can manually verify whether the circumstances in which the facts will be derived are realistic. As an aid rules can then be ordered for example by the number of conditions they have, where rules with more conditions can be considered to place more demands on the business collaboration regarding their application.

Using the above procedure rule steward(s) can check the policies of the BCIM elements they maintain for deficiency. This is useful for the verification of newly defined policies, when derivation rules have been deprecated (or even removed) or when the completeness rules have changed. The rule steward(s) for this purpose verify each policy alternative in a policy conform the outlined steps to identify any missing derivation rules. For example, if the policy alternative of task `estimate repair` does not contain a rule to deduce which actor it must be allocated to, then the alternative is incomplete. Also, pairs of policy alternatives can be verified for deficiencies to determine whether together the alternatives provide complete information. To illustrate, let us suppose that for `Garage Inc`

its `estimate repair`'s policy alternative mandates that `garage repairer` must conduct this task. However, this is not allowed by the policy alternative of `garage repairer`. Then the combination of the two alternatives results in a deficiency. Lastly, the deficiency detection mechanism is useful to test the effects of changes to the completeness rules in a policy alternative on its derivation rules. If it is now required by `Garage Inc` that the task `receive information` must be preceded by another task, then this implies the presence of a new derivation rules. By identifying these as well as other deficiencies using the deficiency detection mechanism organizations can resolve them. Observe that, like for consistency checking, the simulation of designing a business collaborations based on the developed schema can be done to identify missing rules.

6.2 Generating Business Collaborations

In the previous section we discussed the development of design schemas such that they are free of redundancies, inconsistencies, circularities and deficiencies. With the mechanisms for developing such schemas in place we now turn our focus to the generating of designs. In the introduction of this chapter we identified two main issues that need to be resolved in the context of application of rules in order to generate business collaboration designs. We identify three question that require answers: 1) how can we assess whether BCIM elements and their properties, links and attributions are in compliance with the derivation rules in the policies of these elements; that is, how can organizations verify the conformance of designs to the requirements specified in the corresponding design schema; 2) how do we verify that the requirements captured in the resulting designs have validity, alignment and compatibility; that is, how can organizations ensure that the BCIM elements within individual designs are specified and combined in a manner compliant to the control rules in their policies; and 3) how can we generate designs for business collaborations in such a manner that they are consistent while also supporting flexibility and formal adaptability? We address the first two issues in section 6.2.1, where we develop definitions for conformance and validity, alignment and compatibility verification respectively. After that in section 6.2.2 we define a generic algorithm for the design process that facilitates the generating of consistent business collaboration designs based on their design schemas whilst supporting flexibility and formal adaptability.

6.2.1 Conformance, Validity, Alignment And Compatibility

To recall from chapter 1, conformance deals with the question whether the specific requirements of organizations are accurately captured in business collaboration designs. In contrast, validity, alignment and compatibility are concerned with the question whether these designs are complete, correct and consistent. These may appear to be separate issues, however, they are in many ways highly similar to each other in the context of the rule based approach. Specifically, in both cases we are interested in finding out whether business collaboration designs are in compliance with rules. In order for a design to be

conform the derivation rules of the BCIM elements contained in the corresponding design schema, the definitions of these elements must be compliant with these rules. Similarly, if a design is to be consistent, then its BCIM elements must be compliant with the control rules in the policies of these elements. Whether or not a design is compliant with a rule requires a way to interpret and assess the affect of this rule in the context of this design. As such, there is the need to be able to interpret the truth of rules within business collaboration designs, as this determines their meaning. Thus, we need to have some treatment of the meaning of rules in the context of business collaboration design.

The need for such meaning implies the usage of a *semantics of logic* for the Business Collaboration Rule Language. Generally speaking semantics of logic refers to the approaches that logicians have introduced to understand and determine that part of meaning in which they are interested. Several semantics of logic have been proposed over time: proof-theoretic, truth-value, game-theoretical, probabilistic, and model-theoretic semantics. It is outside the scope of this dissertation to treat these in detail. For more details concerning the specific semantics of logic we encourage the reader to the references provided below. Also, before we continue it is important to note that the term 'semantics' here is different from the semantics of the different types of rule as discussed in section 5.3. There we were concerned with the meaning and role of rules within business collaboration. Here we are interested in the meaning of rules in relation to designs.

Having said that, the first semantics of logic, truth-value semantics seek the meaning of propositions in their interpretation. The truth conditions for propositions are given in terms of truth with no appeal to domains whatsoever, hence its name truth-value semantics. Truth-value semantics are also commonly referred to as substitutional quantification, and are typically applied for standard first-order logic. The problem with truth-value semantics in the context of this dissertation is that we do wish to appeal to a specific domain, being the business collaboration models in whose context the rules and policies we intent to employ are to be applied. As such, truth-value semantics are also not suitable for our purposes. Interested readers are referred to (Leblanc, 1976). Game-theoretical semantics (Dybjer, 1997), or game semantics, is another possible approach to the semantics of logic. It is grounded on the concepts of truth or validity on game-theoretic concepts, such as the existence of a winning strategy for a player. Game semantics has been utilized to develop abstract semantic programming languages models and to aid with software verification by software model checking. It is not useful in our work though as we are not interested in finding the best strategies beyond that of prioritization of rules. Probabilistic semantics have been applied to develop probabilistic programming languages. Its semantics has been shown equivalent to and a natural generalization of truth-value semantics, and it thus suffers from the same critique.

A third option is provided by model-theoretic semantics (Hodges, 2006). Such semantics are based on the idea that the meaning of the various parts of the propositions are given by their interpretation in some pre-defined domain. This at first glance fits nicely with the relationship between business collaboration designs and rules as interpreted in the proposed rule based approach. However, typically model-theoretic semantics are employed to assess whether a set of facts satisfies a set of rules. This is different from what we

wish to do in the sense that we intend to combine facts and rules to deduce new facts. In contrast, proof-theoretic semantics does associate the meaning of propositions with the roles that they can play in inferences. Specifically it focuses on the role that the proposition or logical connective plays within the system of inference. Therefore we adopt proof-theoretic semantics to provide semantics of logic for business collaboration rules. Specifically, we discuss how we can utilize the semantics of Datalog with negation for BCRL business collaboration rules. In the sections below we first provide a brief introduction to proof-theoretic semantics and Datalog, and investigate how it can be used to interpret the semantics of derivation and control rules. Then, we define how the obtained semantics can facilitate conformance, and validity, alignment and compatibility verification.

BCRL semantics with Datalog

Proof theory is grounded on the idea of proof theoretic semantics, which is an approach to the semantics of logic that attempts to locate the meaning of propositions and logical connectives not in terms of interpretations (as in model theory approaches to semantics), but in the role that the proposition or logical connective plays within the system of inference. The basic idea is as follows: assume we have a set of rules in the form of formulaes as well as a set of facts. We then combine these two sets by transforming each fact into rules of the form "if 'true', then A"', thus conveying that A is always true (i.e. a rule without conditions). In the context of the rule based approach the formulaes that we wish to combine are derivation and control rules on the one hand, and the modeling description atoms in business collaboration designs constituting facts on the other hand.

Now, at first impression we can simply state that the proof-theoretic truth of a rule depends on whether this rule is instantiated by the facts (that are expressed as atomic formulaes without conditions). That is, given that a rule has certain conditions and conclusions, if in a business collaboration design the conditions of a rule are satisfied, then its conclusion must also be satisfied. This notion of proof-theoretic truth is only sufficient though for standard first order logic. Specifically, because we extend standard FOL with several additions in the BCRL, we require a somewhat more sophisticated notion of proof-theoretic truth than the one introduced above. We need to incorporate the extensions of negation and prioritized conflict handling for non-monotonic rules as well as the usage of numerical, date and time operators (which possible refer to infinite sets). For this purpose we adopt the semantics of Datalog with negation as the foundation for the semantics of BCRL business collaboration rules. Datalog is a rule language for deductive databases and was developed in the late seventies (see e.g. (Ullman and Widom, 1997) for an introduction to deductive databases and Datalog). Using Datalog has the advantage that the semantics of the BCRL rules can be defined in a clear, sound and complete manner. The syntax of BCRL based rules in terms of Datalog is given by the following definitions:

Definition 11
(Vocabulary)

The vocabulary is the set of concepts in each individual design (expressed in terms of the building blocks as defined in the BCIM in section 4.6), that is:

- The set of relation constants with associated arity 'n', being 'element' (E), 'property' (P), 'link' (L), 'attribution' (A) and 'context' (C).
- The set of object constants starting with lowercase symbols, that is, objects from specific business collaboration designs such as `carRepairInformation`.
- The set of variables conveyed by any collection of symbols starting with a capital letter.

□

Definition 12

(Term)

A term is a variable or object constant such as `car repair information`.

□

Definition 13

(Atomic Sentence)

An atomic sentence is a relation of arity 'n' applied to 'n' terms, for example `E(carRepairInformation,resource,agfil-stm)`.

□

Definition 14

(Literal)

A literal is an atomic sentence or the negation of an atomic sentence, such as `¬E(carRepairInformation,resource,agfil-stm)`. If an atomic sentence is not negated, then we can say it is Horn.

□

Definition 15

(Ground Expression)

A ground expression is a literal without any variables. Such expression is called grounded.

□

Definition 16

(Definite Clause)

A definite clause is either an atomic sentence (i.e. fact) or of the form "head ← body"; where the body consists of one or more conjunctive literals and the head of a single, non-negated literal (i.e. an atomic sentence).

□

Both derivation and control rules in the BCRL can be fairly easily expressed as Datalog rules, as their expressive powers are equivalent. For example, the formal rule $R_{disclosurerule}$ from section 5.4.3 can be expressed in Datalog as:

P(ResourceDisclosure,disclosure,true,carRepairInformation, agfil-stm) :-
E(carRepairInformation,resource,agfil-stm) ∧
P(ResourceValue,value,Value,carRepairInformation,agfil-stm) ∧ Value > 500

Here the formal definitions of the different modeling description atoms in $R_{disclosurerule}$ express the various atomic symbols in the rule, whereas the rule as a whole constitutes a definite clause. There are two main issues that require attention when translating from BCRL rules to Datalog definite clauses. A first issue is that in order to have proper semantics we require all definite clauses to be 'safe'. A definite clause is safe in Datalog with negation and arithmetic if (Ullman and Widom, 1997): 1) each variable in its conclusion (i.e. head), 2) each variable in a negated atomic sentence, and 3) each variable in an arithmetic atomic sentence (which is the case for any atomic sentence employing a numerical, math, date or time operator) also appears in a non-negated atomic sentence. To achieve this we test all BCRL rules whether they have proper range restriction; suggested for mathematical operators in e.g. (Topor, 1991). Discussed in among others (Decker, 1987) and (Lloyd and Topor, 1986), the idea behind range restriction is that any variables not adhering to the identified requirements are grounded by replacing them by object constants. Formally we define this as follows:

Definition 17

(Range Restriction)

A variable in a definite clause is properly range restricted if: 1) it occurs in a non-negated, non-arithmetic atomic sentence; or 2) it occurs in a negated and/or arithmetic atomic sentence while also occurring in a non-negated, non-arithmetic atomic sentence.

□

To illustrate the motivation for using range restriction let us take an example. Suppose we have the rule "if ¬p(X), then q(a)". In general then, if the vocabulary contains an infinite number of object constants (as it can in our cases for numbers, dates and times) the falsity of 'p(X)' will be proven an infinite number of times. The same goes for the rule "if price(X) ¡ 0, then buy(a); where 'X' is once again unbounded. To prevent this one option is to simply replace the 'X' in 'p(X)' and 'price(X)' by an object constant. Although the resulting rules will not be logically equivalent to the original ones, they will have a proper range-restricted quantification. However, the more preferred alternative is to warn developers during rule definition already of inappropriately defined variables, so they can then make the necessary adjustments to ensure proper range restriction. For example, in the definite clause "property(tax,TAX,carRepairOperation,agfil-model) :- element(carRepairOperation,operation,agfil-model) AND

property(price,VALUE,carRepairOperation,agfil-model) AND add(VALUE,50,TAX)" all variables are related to those in a non-negated atomic sentence.

It should be noted though that even then, as (Topor, 1991) observes, some resulting definite clauses may not be finite. That is, the interpretation of a range-restricted variable can still be infinite if the number of possible object constants that can be used to ground the corresponding variable in the non-negated atomic sentence is infinite. For example, in the rule "if P(N) AND add(N,1,M), then P(M)" the variable 'M' is range restricted, yet once we add 'P(0)' its interpretation is not finite. Such rules are not a problem though in our approach, since we disallow the definition of cyclic rules involving numerical operators (as described in section 6.1.3). Rather, rules involving numerical, date and time operators will take the form as in the following example definite clause:

property(tax,TAX,carRepairOperation,agfil-model) :-
element(carRepairOperation,operation,agfil-model) AND
property(price,VALUE,carRepairOperation,agfil-model) AND
add(VALUE,50,TAX)

In this rule both variables 'VALUE' and 'TAX' are range restricted in a non-negated atomic sentence. Moreover, their interpretation will be finite in the sense that given the available vocabulary (even if it is infinite) this rule can only be instantiated using very specific grounded relations (due to the presence of one or more textual object constants per relation like `car repair operation`. This sense of finiteness of rules is described in (Topor, 1991) as weak finiteness. Essentially weak finiteness is defined as follows: let a database B be a set of ground atomic formulas (i.e. facts) for the base predicates of a Datalog program P. Also, let T be the set of facts about derived predicates in P that can be inferred from B in at most 'k' applications of the rules in P. If it is then the case that T is finite for all 'k', this implies that P is weakly finite. In relation to our approach P will be the design schema S_d underlying B, where B is the design D. To illustrate, suppose we have in S_d the correctness rule "if element(Operation, operation, Model) AND property(OperationPrice, price, Value, Model), then Value > 0. Here the variable 'Value' in the conclusion could in theory be infinitely grounded, were it not for the fact that it is range restricted by the 'property' relation. And since the variables in this relation can only be grounded using the finite set of object constants (as given by the available facts in the design D), the interpretation of the statement "Value > 0" is finite. This effect is similar as when using typing of variables to avoid infinite interpretations.

A second problem is concerned with control rules that have negation in their conclusion. This is not allowed in Datalog, that is, we can not state "$\neg$B :- A". The work-around we adopt to accommodate the definition of rules like this is to rewrite and represent them in the form "If A and B, then 'problem'". Note that the special 'problem' conclusion can be parameterized to provide additional information concerning the exact nature of the problem (e.g. whether it is an inconsistency, incompleteness or incorrectness). The transformation from the form "If A, then not B" to "If A and not B, then 'inconsistent'" is well known, and is mentioned in (among others) (E. Mayol, 1995). Transformation is

basically done by first ensuring the aforementioned range restriction and then using the transformation procedure described in (Lloyd and Topor, 1984).

In a similar fashion as control rules with negated conclusions we can also rewrite those with positive conclusions. This is useful to obtain uniformity of derivation and control rules concerning their format in Datalog (and their consequent semantics). For example, suppose we have the completeness rule "if there is a resource, then it must have a value". In its current form this rule can not be evaluated in the same way as derivation rules. The reason is that the conclusion of this rule is not a new fact, but rather a constraint on the presence of an existing fact. This can be remedied however by re-stating the rule into "if there is a resource and it does not have a value, then it is incomplete". Semantically this is equivalent, whereas now we can proof that the constraint is not met by proving the 'incompleteness' conclusion. This illustrates that the transformation of non-negated completeness rules is relatively straightforward by 1) adding the conclusion as a conjunctive negated condition, and 2) inserting a special 'conclusion' statement that is parameterized in such way that each of these conclusions is unique (so that every incompleteness can be identified through its proof). The same procedure can be used for the rewriting of consistency rules and correctness rules.

Now that we have defined BCRL rules in terms of Datalog syntax, the question becomes as to what are the semantics of the resulting Datalog rules. Informally speaking in standard Datalog the semantics of rules is based on the models that satisfy those rules. Formally a model is defined as:

Definition 18

(Model)

A model is a set of ground atomic sentences in a language L, where L in this context is a business collaboration design schema S_d.

□

Variables range over all ground terms. That is, given object constants 'a', 'b' and 'c', p(X) means p(a), p(b) and p(c) are true. We can then define satisfaction as (assuming proper quantification of the Datalog rules using explicit universal quantifiers as obtained from the BCRL rule definitions):

Definition 19

(Satisfaction)

Let M be a model and let R be an explicit universally quantified Datalog rule. Satisfaction of such R in relation to M is then defined as:

- $M \models t_1 = t_2$ if and only if t_1 and t_2 are the same term, syntactically.
- $M \models p(t_1,...,t_n)$ if and only if $p(t_1,...,t_n) \in M$.
- $M \models \neg\theta$ if and only if it is not the case that $M \models \theta$, where 'θ' is an atomic sentence in the body B of rule R.

- M $\models \theta_0 \wedge ... \wedge \theta_n$if and only if M $\models \theta_i$ for every i, where 'θ_0' to 'θ_n' are atomic sentences.
- M $\models$ h $\leftarrow$ $b_1 \wedge ... \wedge b_n$ if and only if either it is not the case that M $\models$ $b_1 \wedge ... \wedge b_n$ or M $\models$ or both, where 'b_1' to 'b_n' are literals and 'h' is a non-negated literal.
- M $\models \forall X.\theta(X)$ if and only if M $\models \theta(t)$ for every ground term 't'.

□

If a set of Datalog rules RS has no negation, i.e. when every definite clause in the body of every rule is an atom, it is Horn. Horn rules have a well-defined minimal model, and the semantics for such a set of rules is defined to be that model. That is, the well-defined minimal model satisfies all sentences S in RS in a minimal manner (see for example (Fitting, 2002) for further details). When the rules do include negation (as can be the case for both derivation and control rules), the minimal model is not necessarily well-defined. This is a problem as organizations will prefer that the rules they stipulate (as much as possible) have clear and unambiguous effects. That is, after `Garage Inc` has defined its policies for how to interact with `Lee C.S`, it will prefer that these policies define exactly what to do at any point during the interaction. Thus, we require a way of dealing with negation in such a manner that we obtain a well-defined model, i.e. obtain clear and unique semantics for a set of rules.

For handling Datalog with negation three often employed techniques are stable model semantics (Gelfond and Lifschitz, 1988), well-founded model semantics (van Gelder et al., 1991) and Herbrand semantics with stratified negation. The idea behind stable model semantics is to find so-called stable models that give semantics to the rules without using negation. Stable models are obtained using the Gelfond-Lifschitz transformation (Gelfond and Lifschitz, 1988), entailing two steps in relation to the BCRL: 1) given that a fact 'a' is true, remove all derivation rules that have as its condition '¬a'. These are not applicable anymore, since one of their conditions can no longer be met; and 2) remove all negated conditions from all other derivation rules. The intuition here is that given we did the first step, all negated conditions still remaining must be true (since otherwise they would have been removed already from the rule set).

However, a consequence of interpreting rules with negated statements using stable model semantics is that there can exist multiple, incomparable minimal models (noted e.g. in (Bernstein et al., 2005), (Fitting, 2002) and (Gelfond and Lifschitz, 1988)); this in contrast to non-negated Horn clauses whose interpretation always results in an unique minimal model (also called its canonical model). (Bernstein et al., 2005) additionally notes that not all of the minimal models appropriately capture the intended meaning of rules when negation is concerned. The question therefore becomes how we can distinguish which minimal model is the 'right' model. It is possible of course to defer this question to the developer. However, if there is some other way to determine the minimal model, this would minimize the burden on the developer. One possible solution is given via well-founded semantics, which resorts to using three-valued truth valuations. Basically the idea

is that there are three possible outcomes of a valuation for a rule, being 'true', 'false' and 'undefined'. Rather than assuming the truth of negated statements through lack of proof as in (Gelfond and Lifschitz, 1988), in well-founded semantics explicit proof of their truth is required. If such proof can not be found, then the outcome of the valuation is 'undefined'. The result is that the so-called well founded model at any point in time only contains those things that are necessarily true and necessarily false.

The problem with well-founded semantics in relation to the BCRL is that it requires the possibility of negating rule conclusions in derivation rules. This is needed since otherwise it is not possible to explicitly proof the truth of a negated statement. However, in the BCRL negation in the consequent of a derivation rule is not allowed for. This thus leaves us with the question again whether it is possible to find a minimal model for a set of rules. As mentioned the third possibility for achieving this is the usage of stratification. We described the procedure for stratifying a set of rules in section 6.1.3 already, where we explained how through the construction of a dependency graph any cycles involving negated edges can be detected and consequently resolved by the developer. Here we suffice therefore by stating that we assume that when interpreting the semantics of a rule set, the derivation rules in this set have been statically stratified. The result will then be that these rules have the corresponding unique minimal model as their semantics (see e.g. (Fitting, 2002) and (van Gelder et al., 1991)). We define the semantics that a set of rules entails (i.e. the semantics following from this set) as follows:

Definition 20

(Entailment)

The semantics of a set of stratified rules RS are provided by the unique minimal model M that satisfies their Datalog representations in a minimal manner, that is, RS $\models \theta$ if and only if $\models$M θ; where satisfaction of all rules R $\in$ RS by M is conform Def. 19.

□

With this definition of BCRL semantics in place, another matter we need to deal with is with the semantics of prioritization handling. Since non-monotonic rules assert retractable conclusions, these may at any point be overridden by higher-priority rules. The intuitive meaning of priorities of rules in the context of business collaboration designs is as follows: suppose we have a pair of rules A and B with inconsistent conclusions (where an inconsistency is the case if an applicable consistency rule is not met as discussed in section 6.1.2). Also assume that A has a higher priority than B. If there is then a design such that both the conditions of A and B are satisfied, only A must be part of the design; where if B is part of the design it must be removed. Note that if there is an adequate definition of prioritization for potentially conflicting rules in the rule set, then even though the minimal model can grow and shrink it is always unique and singular in nature (i.e. there is only one minimal model). Moreover, we know that this minimal model gives the desired semantics of the rules as it conforms to the priorities specified by the developers. Formally we state this as:

Definition 21

(Prioritization Semantics)

The semantics of the set of prioritization statements PR for a set of conflicting rules R_1 to R_n in a set of applicable rules RS is provided by the unique minimal model M that satisfies the Datalog representations of R_1 to R_n (conform Def. 19) in which the conclusion of the lowest-priority rule $R_{lowest_priority}$, R_{conc}, is no longer $\in$ M; which effectively is accomplished by stating that $R_{lowest_priority}$ is no longer $\in$ RS (that is, is no longer applicable).

□

Here $R_{lowest_priority}$ is determined through interpretation of the absolute prioritization labels of R_1 to R_n, and the relative prioritization statements in PR (conform the discussion concerning prioritization of the inconsistency detection mechanism for an individual policy alternative described in section 6.1.2). Also note that the above definition covers prioritization of non-monotonic rules. The definition does not apply in case of conflicting monotonic rules. Because monotonic rules can not be invalidated once applied, facts deduced by using them can not be retracted. As such, when two monotonic rules conflict, then user intervention will be required to assess which rule has priority; that is, which is the desired minimal model. Therefore, the handling of conflicts between non-monotonic rules can be pre-defined in terms of priorities and thus the proper semantics can be automatically determined. In contrast, contradictions among monotonic rules must always be resolved via user intervention in order to obtain the desired semantics.

Additionally, observe that Def. 21 is not an augmentation of the normal semantics of Datalog (which does not cater for prioritization and removal of facts). Rather, what happens when a fact is removed due to a prioritization is the following: the fact is removed from the model M by effectively saying that the rule $R_{lowest_priority}$ by which it was entailed is no longer applicable. Rather, the rule with higher priority is now applied. One can consider this as a change in the set of applicable rules RS that is to be satisfied by the model M, where $R_{lowest_priority}$ is no longer considered to be part of RS. As such, for both the old model M_{old} and new model M_{new} entailed by the rule set RS_{old} and RS_{new} respectively, the normal Datalog semantics as stipulated in Def. 20 apply. Consequently, the entailed model M is unique and minimal for both the old and new rule set. Thus, it is not so much that the semantics of the existing rule set RS change (and thus the existing model M), but rather that the rule set RS itself is modified and we therefore obtain a new model M.

Having said, the last issue we need to deal with is the interpretation of modalities. Providing semantics for modalities requires that we somehow reflect the meaning of alethic, deontic, temporal and doxastic modalities. In line with the remarks we made in section 5.2.2 regarding the relation between the modality of a rule and its monoticity and negation, the meaning of modalities is established based on the rule monoticity and clause negation. As such, the above provided definitions on the semantics of monoticity, negation and priorization are applicable. Formal proof concerning these intuitive interpretations of modalities using the monoticity and negation of rules is beyond the scope of this dissertation. For a

starting point on the more complete coverage of the semantics of modalities we refer the reader to (Bull and Segerberg, 1984).

Conformance

Now that we have developed a clear definition of the semantics of BCRL based business collaboration rules in terms of Datalog with negation semantics, we explain how we can use these semantics to assess the conformance of designs to the derivation rules in their design schema. This requires that we have some means of testing whether BCIM elements and their properties, links and attributions in a design are in conformance with their derivation rules. Intuitively this represents the notion that the definition of BCIM elements is compliant with their stipulated derivation rules. Made concrete it implies that BCIM elements are conform their derivation rules if they can be said to satisfy these rules in the meaning as defined in Def. 19. We can thus define conformance as:

Definition 22

(Conformance)

Let a business collaboration design D be a set of arbitrary modeling description atoms $D = (c, e, p, l, a \mid e \in ES \wedge p \in PS \wedge l \in LS \wedge a \in ASwithc.cm = e.em \wedge e.em = p.pm \wedge p.pm = l.lm)$ where ES is a set of elements $\{e_0...e_n\}$, PS a set of properties $\{p_0...p_n\}$, LS a set of links $\{l_0...l_n\}$ and AS a set of attributions $\{a_0...a_n\}$. Also, let DERS be the corresponding set of derivation rules $DERS = \{ders_0, ..., ders_n\}$. Then, D is **conformant**, denoted as D$\cdot$, if $\forall$ ders $\in$ DERS it is true that 'ders' is satisfied by D conform Def. 19.

□

The above definition leaves open as to how to determine the rules that must be satisfied by a design, i.e. how to determine the contents of DERS. The procedure to do so informally is as follows: to recall, derivation rules are part of the policies associated with individual BCIM elements in a design schema. Specifically, each BCIM element has an associated policy comprising a set of mutually exclusive policy alternatives, where each alternative constitutes a set of derivation and control rules. Then, informally speaking the set of rules DERS of a design D is determined by taking the applicable policy alternative in the policy of each BCIM element present in D and adding its derivation rules to DERS. Formally we define the DERS of a design D then as:

Definition 23

(Derivation Rule Set)

Let D be a business collaboration design comprising a set of arbitrary modeling description atoms $D = (c, e, p, l, a \mid e \in ES \wedge p \in PS \wedge l \in LS \wedge a \in ASwithc.cm = e.em \wedge e.em = p.pm \wedge p.pm = l.lm)$ where ES is a set of elements $\{e_0...e_n\}$, PS a set of properties $\{p_0...p_n\}$, LS a set of links $\{l_0...l_n\}$ and AS a set of attributions $\{a_0...a_n\}$. Also let S_d be its design schema. Then, the set of

derivation rules DERS comprising the rule base corresponding to D is defined as DERS=$\{DEPA_{e0} \cap ... \cap DEPA_{en}\}$, where $DEPA_{e0}$ to $DEPA_{en}$ are the policy alternatives that are applicable to e_0 to $e_n \in ES$ as defined in S_d; and where only those rules are taken into consideration for which it is true that they are of type 'computation', 'inference' or 'action enabler'.

□

In combination Def. 22 and Def. 23 enable the verification of conformance of any arbitrary combination of BCIM elements and their properties, links and attributions. As such, it provides a generic mechanism with which the conformance of individual models can be checked for conformance to their model schemas (e.g. a specific message exchange) as well as the conformance of mappings between models in relation to the respective model schemas of these models. For example, `Garage Inc` can use the above definitions to verify the conformance of `car repair information` to its associated policy, of the BCIM elements comprising its operational process to their policies, of the attributions constituting the mapping between this process and its service level counterpart, or the attributions expressing the mapping between the operational process and the corresponding protocol. Observe that alternatively we can rephrase Def. 22 and Def. 23 to state that a design D is conformant if it is true that $\forall$ e $\in$ D it is true that $\forall$ r $\in PA_e$ it is true that D satisfies 'r'; where 'e' is a BCIM element, PA_e the alternative applicable to 'e' from the design schema S_d and 'r' a rule of type 'computation', 'inference' or 'action enabler'.

To exemplify, let us recall from section 5.4.3 the policy of resource `car repair information`. Formally denoted as $P_{carRepairInformation}$ this policy consisted of two mutually exclusive policy alternatives defined as $PA_{carRepairInfoDefault} \oslash PA_{carRepairInfoLeeC.S}$. Let us assume that here we wish to assess the conformance of `car repair information`'s definition with regard to $PA_{carRepairInfoLeeC.S}$. $PA_{carRepairInfoLeeC.S}$, among others, contains the derivation rule $R_{disclosurerule}$ stating that if the resource `car repair information` has value greater than \$500, then disclosure of `car repair information` must be prevented. Formally this was defined in section 5.4.3 as "$\forall$ ResourceValue,Value,ResourceDisclosure [E(carRepairInformation,resource,agfil-stm) $\wedge$
P(ResourceValue,value,Value,carRepairInformation,agfil-stm) $\wedge$ Value $>$ 500 $\rightarrow$
$\Box$ P(ResourceDisclosure,disclosure,true,carRepairInformation, agfil-stm)]".

When performing conformance verification, this rule will be part of the design D that interprets $PA_{carRepairInfoLeeC.S}$. Then, when we further assume that D contains the grounded terms E(carRepairInformation,resource,agfil-stm) and P(carRepairValue,value,600,carRepairInformation,agfil-stm) to describe `car repair information`, it follows that $R_{disclosurerule}$ is a satisfied rule concerning its conditions. This logically entails that it is necessary that P(carRepairValue,disclosure,true,carRepairInformation, agfil-stm) is the case. Following Def. 22 this latter modeling description atom must thus be part of the design D for the specification of `car repair information` to be conform its policy alternative, i.e. it must be necessarily true that 'carRepairInformation' $\models R_{disclosurerule}$. Supposing that $R_{disclosurerule}$ is the only rule in $PA_{carRepairInfoLeeC.S}$, then following Def. 22 it is true that

'carRepairInformation' $\models$ $PA_{carRepairInfoLeeC.S}$, and thus that 'carRepairInformation' $\models$ $P_{carRepairInformation,domainspecific}$.

A slightly more complex example involves more than one BCIM element in the design D. Let us assume that we not only have the facts for `car repair information` in D but also ground terms defining `supply car repair information`. This latter task will have its own policy specified as $P_{supplyCarRepairInformation}$. Let us also suppose that this policy contains one alternative, being $PA_{supplyCarRepairInformationDefault}$, containing the single derivation rule $R_{allocation}$ defined as "∀ Model [E(supplyCarRepairInformation,step,Model) → L(Link, produces,supplyCarRepairInformation,carRepairInformation,Model)]". Then, when we have a combination of `car repair information` and `supply car repair information`, the contents of the Herbrand universe becomes: E(carRepairInformation,resource,agfil-stm), P(carRepairValue,value,600,carRepairInformation,agfil-stm), and E(supplyCarRepairInformation,step,agfil-stm). The set of applicable derivation rules becomes the conjunction of $PA_{carRepairInfoLeeCS}$ and $PA_{supplyCarRepairInformationDefault}$ to which the design D must be conform, comprising the rules $R_{disclosurerule}$ and $R_{allocation}$. This is true if the design D in addition contains the link "L(myLink, produces,supplyCarRepairInformation,carRepairInformation,agfil-stm)", as is logically entailed by $R_{allocation}$.

Validity, Alignment and Compatability

If the BCIM elements in a design are specified and linked in conformance with their policies in the corresponding design schema, organizations can be assured that the design is in compliance with the derivation rules. However, this does not provide indication regarding whether the design is consistent. Therefore, conformance verification needs to be complemented by validity, alignment and compatibility verification to assess that business collaboration designs are properly defined. The reason is that designs can be conform requirements yet be incomplete, incorrect and/or inconsistent in nature. For starters, in a strictly logical sense we typically need the means to prevent the possibility of deducing both 'a' and '¬a' at the same time. However, since we assume that the rules in a business collaboration design schema have been stratified and derivation rules can not have negated conclusions, such logical inconsistency can never occur. The issue is though that logical inconsistencies are but one set of inconsistencies that may occur, there are many more which can not be detected in this fashion. For example, logical inconsistency does not preclude a design to for example define two properties 'price' for the same operation `report estimate`, map tasks to endpoints, define deadlocks or infinite looping behavior, and etceteras, while still being conform requirements. It also does not encompass constraints organizations can have regarding the consistency of their business collaborations. Therefore, it is necessary to extend the logical notion of consistency to a more elaborate form to detect these other varieties of inconsistencies. Additionally, we lack the means at this moment to check whether a design is complete and correct.

The mechanism that we propose to provide organizations with such means is grounded

on the notion of conformance introduced in the previous section. Here though we are interested in the conformance of BCIM elements in designs to the control rules in their policies (as defined in the corresponding design schema). The informal idea is that BCIM elements (and their properties, links and attributions) must be conform the control rules in their policies, because if they are not this means that a problem has been detected. That is, if the conditions of a control rule are true in a design, then the constraint expressed in its conclusion must be met in the design as well. In terms of the Datalog representation of this control rule (where the constraint's conclusion is moved to the conditions and a special 'problem' conclusion is added) this means that the equivalent definitive clause must not be satisfied. In terms of the discussed semantics this means that the modeling description atoms in a design must not satisfy control rules as stipulated in Def. 19. We formalize this notion as:

Definition 24

(Validity, Alignment, Compatibility)

Let a business collaboration design D comprise a set of arbitrary modeling description atoms $D = (c, e, p, l, a \mid e \in ES \wedge p \in PS \wedge l \in LS \wedge a \in ASwithc.cm = e.em \wedge e.em = p.pm \wedge p.pm = l.lm)$ where ES is a set of elements $\{e_0...e_n\}$, PS a set of properties $\{p_0...p_n\}$, LS a set of links $\{l_0...l_n\}$ and AS a set of attributions $\{a_0...a_n\}$. Also, let DCRS be the set of control rules $DCRS = \{dcr_0, ..., dcr_n\}$. Then, D is in conformance with its control rules, denoted as D$\circ$, if $\forall$ dcr $\in$ DCRS it is true that D does not satisfy 'dcr' conform Def. 19.

□

The control rules that make up DCRS are determined in the same fashion as derivation rules in section 6.2.1, that is:

Definition 25

(Control Rule Set)

Let D be a business collaboration design comprising a set of arbitrary modeling description atoms $D = (c, e, p, l, a \mid e \in ES \wedge p \in PS \wedge l \in LS \wedge a \in ASwithc.cm = e.em \wedge e.em = p.pm \wedge p.pm = l.lm)$ where ES is a set of elements $\{e_0...e_n\}$, PS a set of properties $\{p_0...p_n\}$, LS a set of links $\{l_0...l_n\}$ and AS a set of attributions $\{a_0...a_n\}$. Then, the set of control rules DCRS corresponding to D is defined as DCRS=$\{DCPA_{e0} \cap ... \cap DCPA_{en}\}$, where $DCPA_{e0}$ to $DCPA_{en}$ are the policy alternatives that are applicable to e_0 to $e_n \in$ ES (as defined in the corresponding design schema S_d) containing only those rules for which it is true that they are of type 'completeness', 'correctness' or 'consistency'.

□

Like we did in section 6.2.1 we can rephrase Def. 24 and Def. 25 to state that a design D is conformant its control rules if it is true that $\forall$ e $\in$ D it is true that $\forall$ dcr $\in$ PA_e

it is true that D does not satisfy 'dcr'; where 'e' is a BCIM element, PA_e the alternative applicable to 'e' and 'dcr' a rule of type 'constraint' or 'guideline'.

With the help of Def. 24 and Def. 25 we can perform the verification of any combination of modeling description atoms with regard to their conformance with the relevant control rules. To illustrate, if all verified modeling description atoms in Def. 25 are part of the same model M in D and it is true that D does not satisfy any rules in DCRS, then M is valid. This allows an organization to verify models of its business processes, business protocols and business agreements for validity against the corresponding model schemas. If the verified atoms in Def. 25 are from two models M_1 and M_2 describing the same aspect at different levels in D and D satisfies DCRS, then M_1 and M_2 are aligned. Such checking enables an organization to determine whether its models of its business processes, protocols and agreements at different levels are consistent; i.e. whether the corresponding schemas as interpreted in the design D are aligned. Lastly, if the verified modeling descriptions atoms are from two models M_1 and M_2 describing different aspects at the same level in D and D satisfies DCRS, then M_1 and M_2 are compatible. The latter empowers organizations to verify that their business processes support their business protocols and that these protocols meet the requirements agreed upon in their business agreements; i.e. that these processes, protocols and agreements are conform the control rules specified in the corresponding model schemas.

Another interesting property of Def. 24 and Def. 25 is that they can be fine-tuned to verify only the characteristic of completeness, correctness and consistency. If an organization like `Garage Inc` is only interested for example in determining the consistency of a design, then it can easily do so by restricting the control rules to which all the BCIM elements in a design must comply to the set of consistency rules DCCNR (formed by the consistency rules in the applicable alternatives in the policies of those BCIM elements found in the design schema conform Def. 25). Then the set DCRS will only comprise of consistency rules (i.e. DCRS=DCCNR) and consistency can be verified using the above definitions. Individual completeness and correctness verification is facilitated in a similar manner. We will see in section 6.2.2 of Chapter 6 that this allows us to perform the different checks at different times in order to optimize the generating and managing of business collaboration designs.

To illustrate, let us assume the existence of $R_{resourcevalue}$ and $R_{disclosurecontrol}$ in DCRS, defined as "∀ ResourceValue,Value,Resource [P(ResourceValue,value,Value,Resource,Model) → Value > 0] " and "∀ ResourceDisclosure,Value,Resource [P(ResourceDisclosure,disclosure,Value,Resource,Model) → Value=='true']". The first rule is straightforward and conveys that all resource values must be greater than zero. The second rule was defined by `Garage Inc` and depicts that the 'disclosure' property of a resource must always be set to 'true'. Then, when `Garage Inc` verifies the validity of its agreement with `Lee C.S` (which contains `car repair information`'s definition), these two rules become part of the applicable set of control rules. The design D will contain the grounded terms E(carRepairInformation,resource,agfil-stm) and P(carRepairValue,value,600,carRepairInformation,agfil-stm) as well as P(carRepairDisclosure,value,true,carRepairInformation,agfil-stm), which was logically

entailed by $R_{disclosurerule}$. Here, the original conditions of $R_{resourcevalue}$ and $R_{resourcedisclosure}$ become instantiated. Therefore, we evaluate the truth of their original conclusions (that is, the conclusions moved to the conditional part of the definitive clause). Here we can conclude that these conditions are not met; i.e obtain that it is true that `carRepairInformation` does not satisfy $R_{resourcevalue}$ and `carRepairInformation` does not satisfy $R_{resourcedisclouре}$. As such, we can conclude that `carRepairInformation` has been modeled in a valid manner, that is, conform its control rules. Assuming that all other BCIM elements are conform their control rules as well in the agreement (which can be tested in the same manner), we can state that this agreement is valid in relation to its corresponding model schema.

6.2.2 Flexibility And Formal Adaptability

In the previous section we introduced semantics of logics for the derivation and control rules in BCIM element policies using model theory. We then developed definitions with which the conformance of designs to the derivation rules in their design schema can be tested. We also defined when such designs are valid, aligned and compatible in relation to the control rules in this schema. In this section we shall explain how we employ these definitions to generate business collaboration designs in such a manner that they are conform requirements and at the same time are valid, aligned and compatible. We will also demonstrate how the two forms of dynamicity at design time, i.e. flexibility and formal adaptability, are supported. The main idea for generating designs is as follows: we use the definition of conformance as provided in section 6.2.1 in an active manner to generate designs. Concretely, when we wish to derive part of a design we find those rules whose conditions are conform this design. Depending on the specifics of the to-be-designed collaboration different rules will be applied making it possible to support flexibility and formal adaptability. Then, in order to maintain conformance we deduce their conclusions and add them to the design.

The resulting design is then checked for validity, alignment and compatibility using the definitions given in section 6.2.1. As a side note observe that as such control rules indirectly constrain the affect of derivation rules. This makes it possible for organizations to assess whether their design schemas, when interpreted, lead to coherent and valid business collaboration designs, identify which policies are consistent/inconsistent with one another and etceteras. That is, organizations can simulate the design of their business collaborations based on developed design schemas (as mentioned in section 6.1.2). The remainder of this section is structured as follows: we present the algorithm for generating designs, called the *Business Collaboration Design Algorithm (BCDA)*, in section 6.2.2. We next show in section 6.2.2 and 6.2.2 how this algorithm supports flexibility and formal adaptability.

Business Collaboration Design Algorithm

The process of generating designs in a dynamic manner follows a generic algorithm. The requirements for such algorithm are threefold: a first requirement for this algorithm is that it allows business collaboration designs to grow and shrink, where the latter is necessary

as parts of designs may be non-monotonic in nature. Furthermore, the algorithm must respect user indicated preferences as expressed via ranking and prioritization and take the life cycle of rules into account. Finally, the algorithm must ensure that generated designs stay conform requirements (i.e. the derivation rules) as well as not violate any constraints (i.e. the control rules). We call such a design a *well-defined design*. Formally this is stipulated as:

Definition 26

> Let D be a design D, DESR the set of applicable derivation rules and DCSR the set of applicable control rules. Then, if it is true that D is entailed by DESR and D is entailed by DCSR (as stipulated in Def. 22 and Def. 24 respectively), i.e. D$\cdot$ and D$\circ$, then D is a **well-defined design** denoted as D$\odot$.

□

Now, as we generate new parts of a design or modify or remove existing parts the key objective is to ensure that it remains well-defined. That is, a design must move from one well-defined state to another as it is being transformed. We define such transformation as:

Definition 27

> Let T be the transformation operator and D_n a design such that D_n is $D_n\odot$. Then, it must hold that $D_n\odot$ x T $\rightarrow$ $D_{n+1}\odot$, i.e. that the transformation T over a design D_n results in a design D_{n+1} for which it is true that D_{n+1}=$D_{n+1}\odot$ when D_n=$D_n\odot$ was true prior to T.

□

To ensure that transformations of designs move them from one well-defined state to another we employ the definitions introduced in section 6.2.1 to drive and constrain the design process. Specifically, the design process follows the algorithm visualized in Fig. 6.3, which we refer to as the aforementioned Business Collaboration Design Algorithm (BCDA).

In the figure the BCDA is represented in an UML activity diagram style. Steps are denoted by rounded rectangular squares where are connected via arrows. Diamonds represent decision making points, where the conditions of an alternate path are defined within brackets. The all black dot represents the begin point, whereas the white encircled black dot is the endpoint. Fig. 6.3 shows the main steps in the algorithm. The begin situation of the algorithm is one in which it has the design schema S_d of the organization for the to-be-designed business collaboration at its disposal, being: 1) the entire collection of elements EL that may possibly be part of the business collaboration design D (thus all the elements in the different model schemas in S_d); and 2) the set of policies PLC associated with the elements in EL. The algorithm also takes the currently known design D as input. On a high level the algorithm then works as follows:

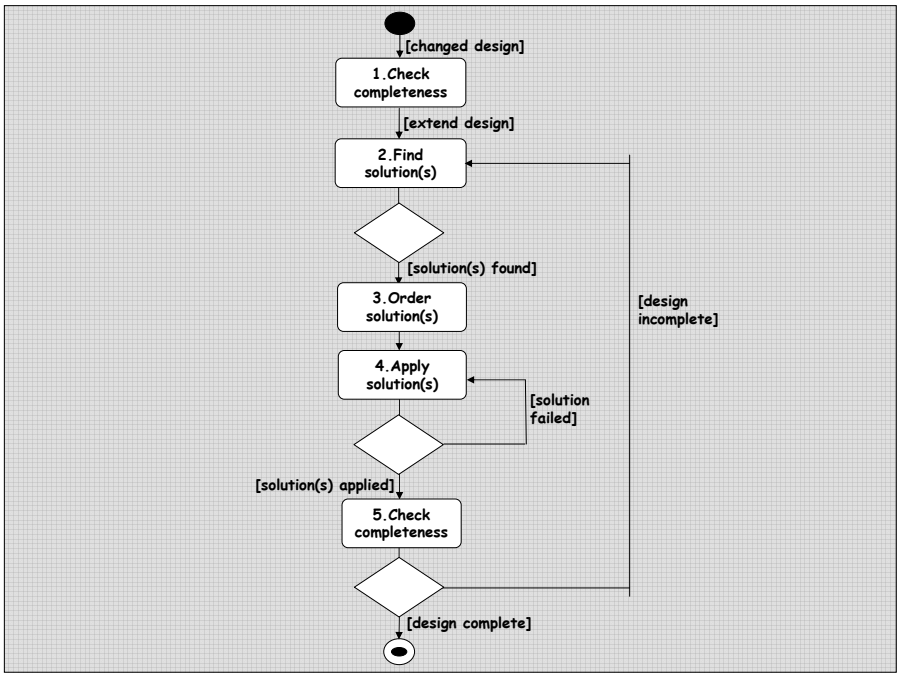

Figure 6.3: Business Collaboration Design Algorithm (BCDA)

1. Every time an expansion of the business collaboration design D is required to further its completion (for example because of a new input message), the algorithm determines which new information needs to be derived to successfully deal with this expansion. To this end it performs a completeness check in **check completeness** by verifying the applicable completeness rules against the current design. If there are any completeness rules that are currently violated, the algorithm sorts them in accordance with their monoticity, rank and relative prioritizations.

2. After that the algorithm attempts for each missing fact to find derivation rules in PLC that can be applied to conclude this fact in **find solution(s)**. If there are no solutions, then the user can intervene by changing the rules (discussed in section 6.3). After that a new attempt is made to find solution(s).

3. When one or more solutions are found, these are sorted in **order solution(s)** by suitability. Optionally, these solutions may lead to different conclusions. In such cases there are multiple strands of reasoning for which solutions are ordered.

4. Next, the top solution on the list is applied to the design in **apply solution(s)** for each strand of reasoning. Solutions are tried in order of decreasing preference until one has been successfully applied. During solution application the derivation of new parts of the design D is checked for correctness and consistency using the relevant correctness and consistency control rules respectively. If no solution could

be applied, then the design D could not be extended in the desired manner. In such cases a change of the rules is required conform the discussion in section 6.3. When the (derivation and/or control) rules have been changed, the algorithm starts in the begin state again.

5. When one or more solutions have been applied, the design uses the completeness control rules in **check completeness** to assess whether all information needed for execution is present. If not, then the algorithm moves back to its begin state to derive this information. Otherwise, it terminates in the end state.

Now, the above five steps provide a high level view of the BCDA. In the following we will give a detailed description of its exact working. In this description we assume that D_n is a well-defined business collaboration design $D_n\cdot$ at time 'n', EL the set of elements as defined in D's design schema S_d), PLC the set of policies associated with these elements, $PAS_{applied}$ the set of policy alternatives chosen to be applied from the policies in PLC, and $DERS_{applied}$ the list of instantiations of applied derivation rules. Also, DCRS is the set of control rules defined here as DCCMR ∩ DCCRR ∩ DCCNR, where DCCMR, DCCRR and DCCNR constitute the sets of completeness, correctness and consistency rules in DCRS respectively. Given the previous, a transformation of $D_n\cdot$ to $D_{n+1}\cdot$ then takes place as follows (where we assume for illustrative purposes that `Garage Inc` has just completed `handle car` in `AGFIL`'s internal business process resulting in the resource `car repair information`, and now wants to derive what step to perform next, how to perform it, and so on):

1. **Check completeness**

   At the start the algorithm only knows that the input design D has changed in some manner and requires expansion. However, unknown is what information is exactly missing. More formally, we started under the premise that D_n was a well-defined business collaboration design $D_n\cdot$ at time 'n', for which it was true that D is not entailed by DCCMR, D is not entailed by DCCRR and D is not entailed by DCCNR. However, due to the change the truth of that D is not entailed by DCCMR can no longer be established. And if D is now entailed by DCCMR, then D will need to be further extended as D is not yet well-defined. To identify the missing facts the algorithm performs a completeness check using the active completeness rules in DCCMR through backward reasoning. Completeness rules are active if they have an 'active' status and are: 1) part of the alternative chosen to be applicable for the element with which the rules are associated; or 2) if no such alternative has been chosen yet part of an alternative whose guard conditions (if any) are met in the current design D. Every such completeness rule whose 'incompleteness' conclusion can then be proven, indicates the presence of an incompleteness. The exact extensions that are then needed to complete D are indicated by those violated completeness rules comprising the list $DI_{to_complete}$. Any missing facts whose addition is known to result in unresolvable inconsistencies (kept track of when checking for and resolving inconsistencies on the list $DI_{facts_leading_to_inconsistency}$) are next excluded

from $DI_{to\_complete}$.

Note that the above means that the completeness rules drive the algorithm to keep extending the design D until it is complete. For example, because the resource `car repair information` has been added, we will need to resolve what step to perform in response, who is responsible for this new step, what temporal constraints are applicable, and etceteras. The order in which these missing facts are to be deduced by the algorithm, is determined by the ranks of the policy alternatives to which each completeness rule belongs. Note that this is only required if both alternatives are part of the same policy of course, since alternative ordering is regarding scope restricted to within individual policies). Also no alternative must have been chosen yet, as otherwise all completeness rules belonging to a policy will be part of the same policy alternative. If this comparison is not conclusive, then the algorithm verifies the monoticity, rank and relative priorities of the violated completeness rules to order them. As such, by giving different priorities to different completeness rules (and their policy alternatives) developers are in the position to control in what manner business collaboration designs are generated.

2. **Find solution(s)**

   Once it is clear which fact has highest priority to be derived, the first action that the algorithm performs is to find out possible answers to the question how the requested extension of the design D can be realized; i.e. which step should follow `handle car` and how this step is to be carried out. This is done in a twofold manner:

   (a) *Determine applicable rules*

   First the algorithm gathers all the derivation rules in the policies in the set of policies PLC (as defined by `Garage Inc`) that are applicable to the current design D to form the set DESR. To this end the algorithm examines each policy P and verifies that if somewhere during design a policy alternative PA has been chosen to be applied for P, then only the derivation rules in PA should be available. If no alternative has been applied yet, then the derivation rules in all the alternatives in P should be added to DERS (that is, for all alternatives whose guard conditions GC are met in the design D, i.e. GC $\in$ D). This ensures that only the possibly applicable derivation rules are at the algorithm's disposal when looking for solutions to extend the design D in the requested manner. In both case the algorithm also takes the status of the policy alternatives and rules into consideration. Only active alternatives and rules are considered as possible candidates. Moreover, as the status of alternatives and rules can change during the design process, the algorithm checks the activation and expiration dates of each alternative and rule and adjusts their status accordingly.

   (b) *Find potential solutions*

The algorithm then attempts to find derivation rules in DERS that enable the derivation of a link from the just completed `handle car` step to another step, where the source role equals 'dependentOn'. Such rules are identified by the algorithm as follows: the algorithm takes the to-be-derived link as its goal and it then attempts to find rules with which it can conclude this goal in the context of the design D. Basically, for each rule 'ders' in DERS the algorithm will assess whether a) its conclusion is the one sought, and b) whether its conditions are true in D, i.e. $ders_{cond} \in D$. If both are true, then 'ders' offers a potential solution. If $ders_{cond} \in D$ is not true, then the algorithm analyzes whether it can prove $ders_{cond}$ using other rules in DERS. If so, then 'ders' in combination with those rules constitutes a possible solution. Note that these rules themselves may have conditions that are proven using yet other rules, as such forming a chain of rules that together lead to the desired conclusion.

(c) *Assess potential solutions*

As a result from the previous the algorithm gathers a set of potential solutions (which as said optionally may constitute multiple derivation rules chained together). Next, the algorithm assesses the suitability of these solutions in relation to the policies that their rules are part of. It does so by verifying whether the to-be-applied rule(s) in each solution are part of policy alternatives PA in such a way that the following is true: firstly, when no alternative PA has been chosen yet for a P (i.e. it is not part yet of $PAS_{applied}$), it must not be the case that a solution uses rules from different alternatives in the same policy P (as alternatives are mutually exclusive). For example, we can not use rules from different alternatives within `handle car`'s policy to deduce what step follows. This ensures that we do not e.g. use rules to derive the 'dependentOn' link, which are part of a policy alternative that is not applicable in the context of the current business collaboration. Also, the alternative to which the used rules belong must be the same as that of the violated completeness rule; since otherwise the incompleteness is resolved with rules from another policy alternative. Finally, when we are deducing a link or attribution, there must be (at least) two solutions such that one solution uses rules from the policy of the source element, and the other solution uses rules from the policy of the target element. That is, the link or attribution must be supported by the policies of both its source and target element. If one of these three requirements is not met, then the solution is not applicable and will thus no longer be considered. Observe that in case no (suitable) solutions could be found, then user intervention is required to change the rules. We return to this matter in section 6.3.

3. **Order solution(s)**

The search for possible ways to deduce information like the 'dependentOn' link typically leads to the identification of one or more solutions. If there is exactly one solution S, then this solution is chosen and subsequently applied (conform step 4

of the BCDA). If there are multiple solutions, then it is possible that they lead to different conclusion(s). For example, it is possible that `Garage Inc`'s rules tell that `handle car` is to be followed by multiple steps conducted in parallel, where one line of reasoning is based on `handle car`'s rules and the other on those of its output `car repair information`. As such we will have multiple solutions with different conclusions. This issue is addressed by applying each of these solutions to the design. When there are multiple solutions leading to exactly the same conclusion(s), we group and then order them so the most optimal one can be selected. 'The most optimal' in this regard is twofold: 1) the algorithm takes into account user preferences as indicated by the ranking and guard conditions of the involved policy alternatives; and 2) the algorithm selects the solution that has the least chance of becoming invalidated later, that is, the solution with which we run the least risk of extending the design D with information that later on has to be retracted. The ordering is done in seven steps:

(a) *Order by rank of policy alternatives*

First, the ranks of the policy alternatives to which the derivation rules involved in the solution belong, are analyzed. Solutions building on rules from higher ranked alternatives will have preference over lower ranked ones. If a solution contains rules from multiple alternatives, then the average of their ranks is used as a measure. Note that only the ranks of alternatives are considered, which are part of policies that have not been previously applied, i.e. are not part of $\text{PAS}_{applied}$. The reason is that the ranks of other alternatives (which are in the set $\text{PAS}_{applied}$) could unwittingly influence the average rank, leading the algorithm to choose a less preferred alternative. For example, suppose we have to order two solutions for a 'dependentOn' link from `handle car` to another step. Also suppose that each solution uses rules from a different alternative within `handle car`'s policy, where the alternative in the first solution is preferred over the one in the second solution (as reflected in their rank). Then, if the second solution also uses rules from another, already applied, high ranked alternative, it would be possible that the second solution is preferred.

(b) *Order by strictness of guards of policy alternatives*

Second, when the first ordering is inconclusive, the algorithm proceeds to order solutions based on the strictness of the guards of the involved policy alternatives. This is determined by counting the number of guard conditions. Preference is then given to the solution with the least number of guard conditions, as this solution will run the least risk in the future of its conditions becoming invalidated. Observe that here the guard conditions of all alternatives involved in a solution are taken into account.

(c) *Order by monoticity and rank of derivation rules*

Third, if the ordering of the policy alternative ranks and guards results in tied solutions, then their rules are taken into consideration. First, the monoticity of the rules necessary to reach the conclusion is checked, where the solutions with

the least non-monotonic rules are ranked highest. Most preferred are thus completely monotonic solutions as these constitute solid knowledge. Solutions that have the same degree of monotonic and non-monotonic rules are then ordered on the basis of the average ranking of their non-monotonic rules, where the rule with the higher average is preferred.

(d) *Order by monoticity of facts*

Fourth, solutions with an equal degree of monoticity of their rules are further ordered by analyzing the facts upon which these rules will be applied. Solutions that built on existing parts of the design D that were deduced via monotonic rules are preferred over those that are (partially) grounded on parts derived from non-monotonic rules.

(e) *Order by number of derivation rules*

Fifth, solutions with the same ranking at this point are then sorted based on the number of rules necessary to deduce the conclusion. The less rules that are needed in a solution, the less chance there is of one of them becoming invalidated leading to retraction of the deduced information.

(f) *Order by strictness of derivation rules*

Sixth, all things being equal at this point solutions are ranked based on the strictness of their rules, where solutions whose rules have fewer conditions are ranked higher. This is done with the purpose of minimizing the risk of such conditions becoming invalidated at some point.

(g) *Order random*

Seventh, in the unlikely event that after this shifting there are still solutions that are ranked the same, then these are randomly ordered.

4. **Apply solution**

After the ordering of solutions has been completed for all sets of distinct solutions, then for each set the solution with the highest rank is applied. Once again, to minimize the risk of having to retract conclusions in the future, application is done in an ordered manner. This order is determined by following the same procedure as described in the previous step. That is, we take the top solutions of each set of distinct solutions, sort them and then apply the highest ranking one. If later on it turns out that this solution can not be applied, then the second best solution in this set of distinct solutions is taken together with the top solutions of the other sets and then ordered. In this manner we minimize the risk that situations in which solutions with different conclusions conflict (e.g. when the price of an operation is set to two different values at the same time by two different solutions) lead to retraction of existing parts of D. The application of a solution S constitutes what we before referred to as a transformation of the design D. That is, given that $D_n\cdot$ at time 'n' the application of the solution S at this time must lead to $D_{n+1}\cdot$. Informally speaking this is achieved by applying all the rules in S, where the application of each rule is

checked for correctness and consistency. If due to incorrectness or inconsistency S as a whole can not be applied, then the algorithm returns to $D_n \cdot$ and attempts to apply the next-ranked solution. If it turns out that no solution can be successfully applied, then user intervention is required to change the rules. We return to this matter in section 6.3.

Application of a solution S is done through application of its derivation rules. However, before actual rule application takes place, the algorithm first verifies that these rules are still applicable. It is possible that during the application of other solutions during extension of D the usage of rules from particular policy alternatives has been ruled out. If so, then the solution S is no longer a viable one. For example, suppose that we have applied a solution based on rules in `handle car`'s policy as well of the policy of its output `car repair information`. Also assume that we wish to apply a second solution to derive a parallel step to be conducted (as described in (2)) that uses rules from `car repair information`'s policy but from a different alternative than the one chosen in the first solution. In this case we can no longer apply the second solution, since only solutions based on rules from a chosen policy alternative may be used.

Now, assuming that the solution S is still feasible, then the algorithm begins with its application. If S consists of a single rule, then this rule will be applied. When S comprises a chain of rules, then the order in which the rules are applied is conform the way they are sequenced in the rule chain (where the rule with which the end goal is derived, is applied last). Regardless, for a to-be-applied rule 'ders' the algorithm performs the following steps:

(a) *Instantiate alternative guards*

If the rule 'ders' that is to be applied belongs to a new alternative PA, that is, PA is not part of $PAS_{applied}$, then the algorithm first attempts to instantiate PA's guard conditions. For any required facts such that they are not yet in the design D, the algorithm tries to derive them following the BCDA procedure. If this fails (e.g. because of inconsistency), then the rule and thus the solution can not be applied. Consequently, the algorithm stops and moves on to apply the next found solution.

(b) *Instantiate derivation rule*

Assuming that PA was already applied or its guard conditions could be instantiated, the algorithm then instantiates the rule 'ders' conditions $ders_{cond}$ on the basis of the design D. All possible instantiations of $ders_{cond}$ are considered and thus the application of 'ders' can potentially lead to multiple instantiations of $ders_{conc}$. Such situations arise when $ders_{conc}$ of 'ders' is a link or attribution, because there the exact instantiation of $ders_{cond}$ affect the instantiation of $ders_{conc}$. For example, it is possible that multiple steps will follow `handle car` and thus more than one 'dependentOn' link is found between `handle car` and other steps.

(c) *Add conclusion*

When verification of the conclusion led to a positive outcome, the algorithm adds the conclusion $ders_{conc}$ to the design D. Formally speaking, the algorithm moves from a situation of $ders_{conc}$ not $\in D_n$ to one with $ders_{conc} \in D_n$. To keep track of what rules have been applied, each instantiation of 'ders' is added to the list $DERS_{applied}$. Observe that the algorithm only adds those conclusions that are necessary to help realize the request extension of D. As such, any other information that may be deduced from the applied derivation rules is ignored.

(d) *Check correctness*

When $ders_{conc}$ has been added to D, next the algorithm determines the set of correctness rules DCCCR. This set will consist of the correctness rules known thus far plus any new correctness rules (if any). This is the case when the rule 'ders' that was applied is from a policy alternative PA in a policy P for which there is not yet a PA $\in PA_{applied}$ (i.e. no alternative was chosen yet for P). If so, then the algorithm takes the correctness rules in PA and adds them to DCCR; where the correctness rules are identified by the 'type' attribute of the rules in PA. Also, as correctness rules like derivation rules can have an activation and expiration date the algorithm will only take those correctness rules into consideration that have a status of 'active' at the moment the check is performed. As a result DCCCR then constitutes the list of active correctness rules.

D is then verified for correctness, i.e. whether D is entailed by DCCCR. If there $\exists$ dcccr $\in$ DCCCR for which it is true that D satisfies 'dcccr', then this means that $ders_{conc}$ constitutes an incorrect fact. Specifically, if there exists a 'dcccr' whose conclusion can be proven, then this correctness rule has been violated. If the violated correctness rule is a constraint (that is, monotonic in nature), then as a result 'ders' can not be applied successfully. In contrast, if the rule is a guideline (i.e. non-monotonic in nature), then 'ders' can still be applied. However, the problem will be noted, so it can be reported to the user at a later time. For example, if a rule would lead us to derive a 'dependentOn' link between `handle car` and `car repair information`, then such link would semantically not make sense and thus should not be added to D. Consequently, the application of the solution as a whole may be compromised if the fact was crucial for its success (i.e. it was in the list S_{facts}). If so, then the solution can no longer be applied in full and it is discarded. Consequently the following solution in line is tried in (3) taking D_n as the starting point. Also, DCCCR is restored to its old state. Note: if the design schema has been checked for redundancy already using the redundancy detection mechanism in section 6.1.1, the algorithm may forego such check here as incorrectness rule conclusions would have been detected and resolved already.

(e) *Check consistency*

After the correctness check the consistency of the design D is verified, i.e. whether D is entailed by DCCNR. All 'dccnr' ∈ DCCNR are applied to see if D satisfies 'dccnr'. Here, as in the previous step, new consistency rules are added if a rule from a PA was applied that is not yet on the list $PA_{applied}$. Also, only active consistency rules are taken into consideration. If it is possible to proof the 'inconsistency' conclusion of a consistency rule (i.e. it is true that D satisfies 'dccnr'), then 'dccnr' is added to the list of detected inconsistencies DI. To exemplify, if we add a 'dependentOn' link between `handle car` and `supply car repair information` when there is also such link from the latter to the former, then there exists a loop that needs to be resolved. If the list DI is empty, then the algorithm continues to (3.e). Otherwise, an attempt is made to resolve the detected inconsistency(ies). First the algorithm orders the list of inconsistencies DI in accordance with the rankings of the associated consistency rules (if present). Then the algorithm removes any inconsistencies known to be unresolvable as present on the list $DI_{failed_resolution}$. Then the algorithm proceeds to resolve each inconsistency in $DI_{ordered}$ as follows:

i. Identify conflicting rules
The modeling description atoms that satisfied the conditions of the violated consistency rule(s) (but not the conclusion) are first identified. Subsequently, the rules that led to their inclusion in D, i.e. those rules with these atoms in their conclusion found on the list $DERS_{applied}$, are singled out. Assuming the just mentioned loop this would involve identifying the rules responsible for the two 'dependentOn' links between `handle car` and `supply car repair information`. Note that it is possible that multiple rules have been applied that led to the same conclusion. This is the case when redundant derivation rules exhibiting a subsumption relationship are contained in DERS, since then two or more rules will be applied to deduce the same conclusion. It is also the case for derived links and attributions, since such facts require that rules associated with the source and target element are involved. In such cases during the analysis of the inconsistency (described in the next step) the 'worst' rule for each instantiating atom in the inconsistency is selected in accordance with the ordering of rules described in steps (2.c) to (2.g).

ii. Analyze priorities
After that the available prioritization information for the conflicting rules is analyzed. The conclusion of the rule with the lowest priority is then selected for removal in order to resolve the inconsistency. If no conclusive prioritization information is available, then user intervention is required. This is the case if the violated consistency rule is monotonic (i.e. a constraint rather than a guideline) and: a) all involved derivation rules are monotonic, since in principle the conclusions of such rules can not be retracted; b) some or all derivation rules are non-monotonic, but their rankings are the same and the

prioritization statements are absent or inconclusive; or c) the inconsistent conclusions have been deduced on the basis of the same derivation rule. If there is no conclusive prioritization but the violated consistency rule is a guideline, then the algorithm will not attempt to resolve the inconsistency. Rather, it will only place the violation on the list $DI_{failed_resolution}$. To illustrate, if the newly applied derivation rule for deriving the 'dependentOn' link has a higher rank than the old derivation rule, then the old 'dependentOn' link from `supply car repair information` to `handle car` must be removed. Note that we do not explicitly consider modalities here as we assume that these have been accurately reflected in the monoticity, ranking and relative prioritization of the derivation and control rules (conform the discussion regarding this matter in section 5.2.2).

iii. Resolve inconsistency
If the to-be-removed conclusion $ders_{conc}$ is the one that was to be added, it is possible that the solution as a whole can not be applied anymore. This is the case if $ders_{conc}$ is crucial for the solution as a whole, which is e.g. the case for the 'dependentOn' link. If so, then the solution is abandoned and another solution is tried taking D_n as a starting point again in (3). Morever, the algorithm reverts $DERS_{applied}$ and DCCCR back to their previous state. Finally, the conclusion $ders_{conc}$ will be added to the list $DI_{facts_leading_to_inconsistency}$, which is used during completeness checking to ensure that we do not attempt to deduce facts known to lead to inconsistency. If on the other hand an existing part of the design D is removed, then a more complex procedure is followed:

A. First the algorithm checks the rules in the solution S currently being applied to assess whether removal of the selected conclusion $ders_{conc}$ does not invalidate the guard conditions of the involved policy alternatives or makes it impossible for other rules in S to be applied. If this is the case, then the algorithm verifies whether the violated consistency rule is a constraint or guideline. If it is a constraint and thus it must be met, then $ders_{conc}$ must be removed and as a result S can not be completely applied in a successful manner. Consequently the algorithm returns to the beginning of step (4) to apply the next solution in line to go from D_n to D_{n+1}. If it is a guideline, then the algorithm makes note of it and does not attempt to resolve the inconsistency. Rather, the algorithm puts this inconsistency on the list $DI_{failed_resolution}$. In case $ders_{conc}$ is not crucial to the solution as a whole, then the algorithm simply continues by deleting the to-be-removed conclusion $ders_{conc}$ from D, thus the existing 'dependentOn' link from `supply car repair information` to `handle car` would be removed. After that the rule instantiation(s) with which the conclusion was derived, are removed from the list $DERS_{applied}$. Simultaneously, an assertion is added to D expressing that the removed

conclusion used to be part of D. This assertion is expressed using temporal modality 'P', i.e stating that 'it was the case that the conclusion was true'. This allows tracing what parts of the design D were undone at some point, which is useful for example for analysis of carried out business collaborations.

B. The addition of the just mentioned assertion is also done, because removal of an existing part can trigger the application of rules that specify what to do in the event that a part of D is removed. Therefore, after removal of a conclusion the algorithm looks for rules in DERS that as a result have become instantiated conform step (2). If there are any, then these are subsequently ordered and applied following the same procedure as described thus far in steps (3) and (4). For example, if due to the addition of the 'dependentOn' link to the design D an earlier performed step `process car` needs to be retracted, then it is possible that this prompts the application of a rule that adds a compensatory step to D. If during the procedure an incorrectness occurs or inconsistency occurs that can not be solved, this means that the procedure to handle the retraction of ders_{conc} has failed and subsequently the next one (if present) is tried.

C. When an existing part of the design D is removed, the design may become inconsistent in the sense that other parts are invalidated. If this happens, then appropriate action must be taken to remove such parts as well. For this purpose the algorithm performs a two-fold verification: 1) first it checks for each policy alternative in $\text{PAS}_{applied}$ if its guard conditions (if any) are still met. If this is not the case any more for an alternative PA, then this means that application of all of its derivation rules must be undone. The algorithm therefore checks the list $\text{DERS}_{applied}$ to find such rules and subsequently removes them following the procedure in step (4.d.iii.E). PA is also removed from the list $\text{PAS}_{applied}$. Moreover, the control rules from PA in DCRS are removed as well. Alternatively, if the guard conditions of PA are still true, then each of its applied rules 'ders' is verified. Concretely, the algorithm checks for each 'ders' if it is true for its instantiations that the removed part is part of ders_{cond}. If so, then this means that the conditions of this rule have been violated and thus that this conclusion needs to be removed. To exemplify, if `process car` is removed, then the rule that linked it to `garage owner` is no longer satisfied.

D. The algorithm handles a violated rule by first verifying whether the violated derivation rule 'ders' is monotonic or non-monotonic in nature. If 'ders' is non-mononotic, then its conclusion is itself removed from D by carrying out the activities described in step (4.d.iii). If the conditions of an applied monotonic rule have been violated,

then an error is issued to encourage user intervention. The reason is that, in contrast to non-monotonic rules, the knowledge deduced from monotonic rules is regarded to be irrefutable. As such, it is not possible to automatically undo application of a monotonic rule. Rather, this is left to the user. The user can then opt to remove the rule's conclusion after all by carrying out step (4.d.iii). Alternatively, the user may modify the rule such that its conditions are no longer violated (discussed in section 6.3). Note that in case the violated consistency rule is a guideline, the algorithm will not request user intervention. Rather, the algorithm will place the inconsistency on the list $\mathrm{DI}_{failed_resolution}$ and continue on to the next to-be-resolved inconsistency.

If the user opts to remove the rule's conclusion, or when the rule is non-monotonic in nature, the instantiation of 'ders' in $\mathrm{DERS}_{applied}$ is removed; for example the instantiation that led to the aforementioned link between `process car` and `garage owner`. After that the algorithm determines whether the conclusion in 'ders's instantiation is supported by rules other than the one just removed (possible only, as mentioned, if DERS contains derivation rules subsuming one another). If this is the case, then there is no need to remove ders_{conc} itself from D, since a different line of reasoning still underlies ders_{conc}. Consequently, the only action is to remove the violated instantiation of 'ders' from $\mathrm{DERS}_{applied}$. Otherwise, ders_{conc} must be removed from D, which in turn may possibly lead to further removal again until all violated parts of design D have been retracted.

Note that in case an instantiation of a rule that supports a link or attribution is removed, then it must remain true that this link/attribution is supported by instantiations of a rule in the policies of the linked elements. If this is not the case, then ders_{conc} must be removed from the design D (as support is then lacking in D). In all cases that the fact ders_{conc} is removed, it is placed on the list $\mathrm{DI}_{facts_leading_to_inconsistency}$ to avoid that we attempt to deduce it later again. Finally, when a link is removed a check is to be done concerning the relations of its source and target element with other elements in the model. If no such relations exist anymore due to the removal, then the source/target element is no longer effectively part of the design D. As such, it must be removed and any rules applied based on its presence undone. To illustrate, if the step `process car` is no longer connected to any other step (the case when removing all of its 'dependent on' and 'dependent off' links), then all information about this step must be removed and decisions made based on this information undone.

E. When at some point all the violated policy alternatives and/or rules have been handled, the consistency of D is checked again. The retraction of information may have left D in an inconsistent state, there might be inconsistencies that still need to be resolved, or other inconsistencies can be resolved as well. This results in a new list $DI_{ordered}$. The algorithm next compares this list to $DI_{failed_resolution}$ and removes any (guideline) violations it attempted already to resolve. If there are then remaining inconsistencies, the algorithm returns to the beginning of step (4.d) to address them. If this is not the case, then the algorithm continues on to step (4.g). The reason for this somewhat elaborate procedure is that otherwise the algorithm will keep on detecting the same violated guidelines and subsequently attempt to resolve them; where earlier attempts to do this have failed already because of lack of available prioritization information.

(f) *Update rules*

During the course of the application of a solution, derivation rules may be applied that stem from policies P for which no applicable alternative PA has been chosen yet, i.e. there is no PA yet for P in $PAS_{applied}$. If this is the case, then application of a derivation rule R from PA means that this alternative will be the one applicable to the design D. As such, in the future only derivation rules from PA may be used to further extend D. To ensure this we add the PA of R to $PAS_{applied}$ once R has been applied. The effect is that in future cycles of the algorithm only solutions using derivation rules from PA will be approved when finding solutions in (1). This also ensures that only the control rules present in PA will be used to constrain the design D. For example, when we use a rule from `handle car`'s default policy alternative to derive the 'dependentOn' link, then this prohibits future usage of derivation and control rules in its other alternatives.

5. **Check completeness**

When we have successfully applied a solution to the design D, we conclude by checking for completeness again. The idea is that at this point the derived information has become part of the design in a correct and consistent manner. Undetermined however is whether D is also complete, that is, whether the business collaboration can continue. In the example D would be complete if we have all the information needed to carry out the step after `handle car` like who is responsible for it, which tasks it requires at operational level, and any other completeness requirements that have been stipulated. To recall, we started under the premise that D_n was a well-defined business collaboration design $D_n\cdot$ at time 'n', for which it was true that D is not entailed by DCCMR, D is not entailed by DCCRR and D is not entailed by DCCNR. The truth of that D is not entailed by DCCRR and D is not entailed by DCCNR is checked in (3.a.ii) and (3.a.iii) for every addition to D. However, the fal-

sity of whether D is entailed by DCCMR has of yet not been established. If this can not be proven, then D will need to be further extended as it is not yet well-defined. In such case the algorithm will move back the begin state to further extend D via the described steps until at some point D is no longer entailed by DCCMR again. Note that also here DCCMR will be updated by the algorithm in case new alternatives were applied (i.e. not on the list $\text{PA}_{applied}$) as part of the solution. Also, only active completeness rules are used in the completeness check. Furthermore, missing facts on the list $\text{DI}_{facts_leading_to_inconsistency}$ are excluded from the completeness check results.

The exact extensions that are needed to complete D are indicated by the violated completeness rules comprising the list $\text{DI}_{to_complete}$. In this sense the completeness rules drive the algorithm to keep extending the design D until it is complete. For example, after we have added the 'dependentOn' link between `handle car` and `supply car repair information` we will need to resolve who is responsible for this new step, what temporal constraints are applicable, and etceteras. These issues will be identified by the completeness rules. As a result the algorithm will then attempt to address these issues; where after each extension a new completeness check will be performed. Observe that if the violated completeness rules are monotonic (i.e. constraints) the algorithm will request user intervention if the required information can not be derived. If however a completeness guideline was violated that the algorithm was unable to resolve, it makes note of the violation via the list $\text{DI}_{failed_complete}$ and then continues. The algorithm uses this list after each completeness check and compares it to the new list of completeness errors $\text{DI}_{to_complete}$. When all errors in $\text{DI}_{to_complete}$ are part of $\text{DI}_{failed_complete}$, then the algorithm does not attempt to further extend the design. The reason is that the algorithm already tried to resolve the problems before and failed. As such, because the design D will not change anymore (since no incompleteness can be resolved), there is no point in continuing.

A final remark concerning the previous concerns the order in which violations are resolved. Important to realize is that this order is not mandated by the BCDA. Rather, such order is imposed through appropriate sequencing of the completeness rules by ordering them using their rank (as also mentioned in the first step of the algorithm). For example, we may prefer to first resolve any incompleteness of strategic models in the design D before mapping them to the operational level. However, regardless of the order in which completeness problems are resolve, once D is no longer entailed by DCCMR, then D is well-defined and thus it is true for D that it is D·. Consequently, the algorithm finishes in the end state as D has undergone a transformation moving from one well-defined state to another. As such, the carrying out of the newly derived and/or modified parts of the design D can now take place.

This concludes the discussion of the Business Collaboration Design Algorithm for the generating of business collaboration designs. In the following two sections we will demonstrate how this algorithm supports flexibility and formal adaptability, that is, how changes can be incorporated when generating designs.

Flexibility

Flexibility, to recall, constitutes the ability with which business collaboration designs can adapt to predictable changes. Traditionally, this type of change assumes the existence of a pre-defined design in which either points of dynamicity have been built in or in which partial information has been specified that is completed at runtime. As we have seen in the context of the rule based approach such explicit design does not exist. Rather, the policies and rules associated with the different BCIM elements in the design schema of a business collaboration constitute the design in a sense, but with the important difference that they can lead to multiple unique designs. In other words, in the rule based approach no definite information is specified at design time and the design is derived at runtime in an incremental manner by application of the derivation rules. Consequently, each design will be completely unique and tailored to the specific circumstances of the individual business collaboration. As such, the ability to handle predictable changes is realized through the appropriate definition of policy alternatives and their derivation rules, which will steer design in one direction or another; where the BCDA effectuates newly defined alternatives and/or derivation rules into the designs. The corresponding control rules (if any) will be used by the BCDA to ensure the completeness, correctness and consistency of the resulting designs.

To illustrate, let us look for a moment at the private behavior of `garage repairer`. When `garage repairer` has performed `estimate repair`, there are two possible ways to proceed. If the repair cost in `car repair report` is estimated to be below $500, `repair car` is initiated. Otherwise `get approval` is first carried out in order to obtain approval for the repair. To facilitate this dynamic behavior during execution `Garage Inc` specifies two policy alternatives in the policy of `car repair report`: one describing what to do if the estimate is below $500 and one what to do if the estimate is higher. The alternatives themselves will contain the appropriate rules to express what to do in each situation. Then, at runtime when the design is generated a choice will be made depending on the actual value of the car repair estimate in `car repair report`. Concretely, the BCDA will find a solution to the question as to how to handle `car repair report` that is based on rules from one of the two alternatives. Another example is that `Lee C.S` dynamically determines how `report estimate` in `claim management service`'s protocol is mapped to an internal operation on the basis of the time at which the incoming message is received to achieve load balancing. The consistency, correctness and completeness of the design in both examples is ensured through the checks that are performed by the BCDA.

Formal Adaptability

The second form of dynamicity, formal adaptability, represents the ability with which business collaboration designs can handle unpredictable changes at runtime. By convention it seems such changes are regarded exceptional in nature. That is, they are events that are known to happen, but which are considered to deviate from the normal behavior. However, as we argued in section 1.1.2 of Chapter 1 already such distinction typically

seems to be made on the basis of subjective preferences. It is therefore beneficial in the rule based approach we feel that this of type of distinction is unnecessary as designs are generated based on the circumstances of the business collaborations. Whether or not these circumstances are considered to be exceptional or not does not matter in the sense that it does not affect the way in which they are handled. In both cases, just as discussed for flexibility, suitable policies and rules can be defined, which prescribe what to do in each situation (be they considered to be 'normal' or 'exceptional' in nature). Then, the BCDA ensures that the appropriate derivation rules are applied when developing new designs.

To demonstrate, if we examine the behavior of `garage repairer` again, there might be occasions on which `garage repairer` is not sure about the car repair estimate he/she made. This may be interpreted by `Garage Inc` as an exceptional event as under normal circumstances `garage repairer` would be capable of making such an estimate. However, in the context of the rule based approach this does not matter. Rather, `Garage Inc` simply includes another policy alternative in the policy for `car repair report` that ensures that if no estimate could be made `estimate repair` must be followed by the task `get external second opinion`. The rule enforcing this would accordingly state something like "if the estimate is unknown, then get a second opinion from an external expert". If such 'exception' then unexpectedly occurs at runtime, the design will be generated accordingly to handle this situation. As such, there is no difference in the manner in which flexibility and formal adaptability are supported. Therefore, there is no need to introduce artificial constructs such as exception/event handlers that lack real life semantics. Rather, rules perform these functions in the manner as just described. Once again, the consistency, correctness and completeness of the business collaboration design is ensured by the checks carried out in the BCDA using the (re-)defined control rules. Note that the above example implies of course that `Garage Inc` has developed (or will develop) an agreement with another party that will perform the required external second opinion.

6.3 Managing Business Collaborations

In section 6.2.2 we explained how designs can be generated at runtime through the application of rules in order to carry out business collaborations. We also demonstrated how the resulting BCDA algorithm facilitates flexibility and formal adaptability during the design process. In this discussion we made the assumption though that we were generating new designs. As such, we stayed clear from making statements about the role of dynamicity in already running business collaborations; that is, we did not explain how dynamism and undefined adaptability can be accommodated. Generally speaking both types of change are handled in the rule based approach through the definition of new rules (possible for new elements) and modification of existing rules in the design schema of the to-be-modified business collaboration design. Changes to the rules may however compromise the completeness, correctness and consistency of the design. Because of the highly volatile nature of these rules, a change can quickly lead to duplications of rules, inconsistency of rules, falsely pursued business goals, and so on. In such situations the very solution that was to

enable us to make business collaboration dynamic turns against us; and in fact becomes a liability leading to situations in which organizations are led to make bad decisions costing them money and potentially business opportunities. To avoid such potentially damaging situations mechanisms are needed with which the impact of a change to the rules of a business collaboration can be assessed.

The rule management mechanisms described in section 6.1 allow organizations to change the design schemas of the business collaborations in which they are involved. Concretely they allow the modification of the rules in the policies of BCIM elements whilst ensuring that the derivation rules remain in conformance with the control rules. As such, the organizations in a business collaboration can be assured that their design schemas remain complete, correct and consistent with regard to themselves and each other. However, the mechanisms do not provide the means to incorporate the effects of modifications in existing business collaboration designs, which are based on the modified design schema. That is, they do not give support for dynamism and undefined adaptability. In this section we address this issue through the development of a procedure with which business collaboration designs can be updated when their underlying design schemas change. This procedure, called the *Business Collaboration Management Algorithm (BCMA)* is presented in section 6.3.1. We then show this algorithm supports dynamism and undefined adaptability in sections 6.3.2 and 6.3.3 respectively.

6.3.1 Business Collaboration Management Algorithm

Given that a design schema consists of BCIM elements and their policies, two main types of change can be made: 1) the addition and removal of BCIM elements from the schema; and 2) the modification of the policies of existing BCIM elements in the schema. In relation to the requirements with regard to dynamicity this must be accommodated in a procedure in such manner that support is offered for both dynamism and undefined adaptability. At the same time this procedure must guarantee that no (potentially modified) control rules are violated, that is, that the modified designs remain complete, consistent and correct. Rephrased we state this as that a design must move from one well-defined state to another when it is changed. We define such change as:

Definition 28

> Let C be the change operator and D_n a design such that D_n is $D_n\odot$. Then, it must hold that $D_n\odot$ x C $\rightarrow$ $D_{n+1}\odot$, i.e. that the change C over a design D_n results in a design D_{n+1} for which it is true that D_{n+1}=$D_{n+1}\odot$ when D_n=$D_n\odot$ was true prior to C.

□

To ensure that changed designs move them from one well-defined state to another we define a procedure for existing business collaboration designs that builds on the Business

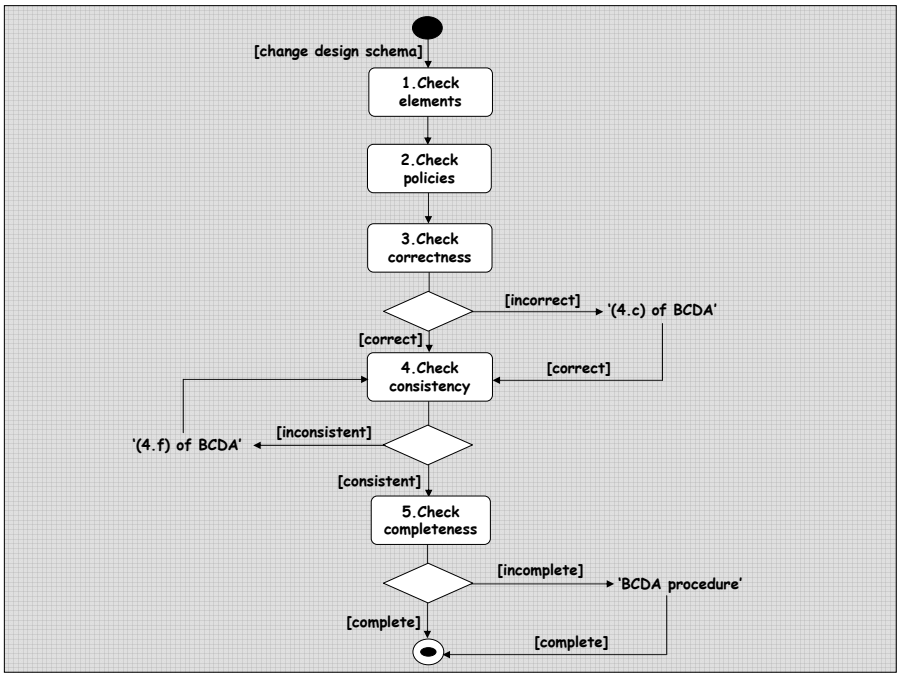

Figure 6.4: Business Collaboration Management Algorithm (BCMA)

Collaboration Design Algorithm described in section 6.2.2. This procedure, the aforementioned *Business Collaboration Management Algorithm (BCMA)*, is visualized in Fig. 6.4.

As can be seen in the figure the BCMA takes the elements EL_{old} in the current design schema S_{old} and their policies PLC_{old} as input together with those in the new design schema S_{new}, being EL_{new} and policies PLC_{new}, and the current design D. Then, at high level the BCMA incorporates changes to EL_{old} and PLC $_{old}$ into the design D by performing five steps:

1. First the algorithm checks the schema S_{new} for changes, such as that new elements have been added or existing elements have been removed by comparing it to S_{old}. The algorithm subsequently effectuates the effects of found discrepancies between EL_{old} and EL_{new} into the design D, where it also updates PLC_{old}.

2. For all elements that are present in both EL_{old} and EL_{new}, the algorithm compares their policies for changes like the addition, modification and/or removal of alternatives and derivation rules. The algorithm then assesses the impact of found modifications on D and updates D accordingly. PLC_{old} is updated as well.

3. Once all changes have been processed, the algorithm next verifies the correctness of the resulting D (using possible modified correctness rules from PLC_{new}). If any problems are found, then these are resolved as described in (4.c) of the BCDA after which consistency is verified again.

4. After that the algorithm verifies the consistency of the resulting D using the consistency rules in PLC_{new}. If any inconsistencies are found, then these are resolved as described in (4.f) of the BCDA. Subsequently the consistency of D is verified again until no more inconsistencies can be found.

5. When D is consistent, the algorithm verifies the completeness of the design. Due to the modification of EL and PLC parts of the design may have been retracted, leaving it in an incomplete state. These problems must first be resolved before the business collaboration described in D can continue. To this end the procedure of the BCDA is followed, but now with the updated design D, elements E and policies PLC.

Now, the above steps describe the workings of the BCMA in a high level manner. In the following we will give a detailed description of the algorithm. We assume that D_n is a (partially) well-defined business collaboration design $D_n\cdot$ at time 'n', EL_{old} the current set of all relevant elements for D in design schema S_{old}, PLC_{old} the set of policies associated with the elements in EL_{old}, $PAS_{applied}$ the current set of policy alternatives chosen to be applied from PLC_{old}, and $DSR_{applied}$ the list of current instantiations of rules. Similarly, S_{new} is the new design schema comprising EL_{new} and PLC_{new}. Also, DCRS is the set of control rules in which DCCMR, DCCCR, and DCCNR constitute the sets of currently applicable completeness, correctness and consistency rules respectively. The BCMA then resolves any differences between EL_{old} and EL_{new}, and between PLC_{old} and PLC_{new} as follows:

1. *Check elements*

   To begin with, the available elements in the design schema underlying the design D, may have changed. Therefore, the algorithm compares EL_{old} and EL_{new} in three steps:

   (a) Element added

   First the algorithm checks whether a new element has been added to the schema S_{new} compared to S_{old}, that is, there is an element 'el' $\in EL_{new}$ such that it is not the case that 'el' $\in EL_{old}$. When this is the case, then basically another building block becomes available in the schema underlying the design. Concurrently, the policy P of the new element 'el' will be added to PLC_{old}. Such change has no further direct impact on the existing design D. Rather, the element and its policy can be used during future cycles of the BCDA. For example, `Garage Inc` may decide that an additional step `update administration` is required in its strategic process to update the administration. Such step may then be added during design, e.g. while resolving incompleteness in the modified design in step (4).

   (b) Element removed

   The algorithm next verifies whether an element has been removed from design schema S_{new} compared to S_{old} (e.g. because it is considered to be no longer

relevant for D), i.e. there is an element 'el' such that 'el' $\in$ EL_{old} but not 'el' $\in$ EL_{new}. If in such case the removed element was already part of D, then this means that part of D must be retracted. The procedure followed in this regard is identical to the one described in step (4.f.iii) of the BCDA (with the exception that here we do not have to check whether removal invalidates a to-be-applied solution). In addition, removal of the element from EL also means removal of its policy in PLC_{old}. As such, any rules in this policy that were applied already have to be undone. This is done as described in step (4.f.iii.E) of the BCDA, where all instantiations of the applied derivation rules are removed from $DERS_{applied}$. In addition, the element removal will lead to the removal of any control rules found in DCRS that were part of the applied alternative. An illustration is that `Garage Inc` no longer supplies car repair information to `Lee C.S`. This means that `supply car repair information` will no longer be included in designs. If this is also to be effective for a currently running business collaboration containing this step, then it must be removed from its design D and the associated derivation and control rules must be updated (as these are no longer relevant for D).

(c) Element modified

After that, the algorithm compares BCIM elements that are both in EL_{old} and EL_{new} to see if there are differences, together comprising the list EL_{same}. For each of the BCIM elements the algorithm compares their old and new policies conform the procedure described in step (2).

2. *Check policies*

Once the algorithm has checked D's new design schema S_{new} in comparison to the old one S_{old}, it then continues to verify the policies associated with the elements that are present in both the old and new schema (i.e. on the list EL_{same}). Concretely, the algorithm assesses whether: 1) the alternatives of policies have changed; and 2) the derivation rules inside policy alternatives have changed. The followed procedure consists of the following steps:

(a) Check alternatives

For each policy pair P_{old} and P_{new} the algorithm checks whether P's alternatives have changed as follows:

i. The algorithm first assesses whether new alternatives PA have been added to P, that is, PA $\in$ P_{new} and not PA $\in$ P_{old} (e.g. to specify how to handle a thus far not yet encountered situation). Such change is handled by simply incorporating the new alternatives, after which their derivation and control rules are available for usage during future design. An example is that `Garage Inc` receives an old-timer that needs repair. Supposing that this has not happened before, `Garage Inc` can add a new alternative to the policy of `car information` to dictate that if a to-be-repaired car is an old-timer, an

external expert must be hired to oversee the repair. Any associated control rules, e.g. to constrain what expert may be hired, can be included in this alternative as well.

ii. After that, the algorithm determines whether there are any applied alternatives PA (PA is part of $PAS_{applied}$) such that PA $\in P_{old}$, but not PA $\in P_{new}$. For each applied PA that has been removed, the algorithm undoes any applied derivation rules following the procedure in step (4.f.iii.D) of the BCDA. The algorithm also removes the control rules in PA from DCRS. Otherwise, such removal does not have a direct impact on D, but rather only on future design. To demonstrate, if `Garage Inc` decides to no longer make an exception for old-timer repair (e.g. because they have acquired in-house expertise), then the old-timer policy alternative needs to be removed from `car information` and any rules applied in this alternative be undone (e.g. pertaining the hiring of an external expert to look at the old-timer). Unapplied alternatives that have been removed, do not impact the design D. The algorithm does update PLC accordingly, that is, removes the derivation and control rules in the removed policy alternatives from PLC.

iii. Lastly, for any alternatives PA $\in P_{old}$ and PA $\in P_{new}$ the algorithm verifies whether this existing PA has been modified. Specifically, it checks whether PA's guard conditions GC and/or its status have changed. If PA has not yet been applied, i.e. is not on the list $PAS_{applied}$, then these changes do not impact the current design D. If however PA is part of $PAS_{applied}$, then D might be affected. In case the guard conditions GC were changed, re-evaluation of their truth is required. When the result is that it is no longer true that GC $\in$ D, any rules in PA that were applied (on the list $DERS_{applied}$) are undone. The same is true if the activation and expiration date of PA have been changed such that the alternative was not applicable at the time it was applied (i.e. did not have an 'active' status). Each rule is undone as described in (3.f.iii.E) of the BCDA. Also, the control rules of PA in DCRS are removed. For example, suppose that the new old-timer policy alternative was implemented by `Garage Inc`, but now the definition of 'old-timer' has changed. Then, in some situations it may have been incorrect to contact an external expert and this must be undone. Alternatively, the new old-timer policy alternative may have been applied too soon due to an incorrectly defined activation date. In such situation application of the alternative's derivation rules must also be undone. Moreover, any associated control rules have now become irrelevant and thus must be removed from PLC as well. For each PA that has not been modified or in such manner that it is still applicable, the algorithm continues to step (2.b) to check PA's derivation rules for changes.

(b) Check rules

When an existing alternative PA is still applicable, the algorithm verifies its

derivation rules for modifications as follows:

i. The algorithm starts by determining if there are any derivation rules 'ders' such that 'ders' $\in$ PA_{new} and not 'ders' $\in$ PA_{old}. Such rules are incorporated in PLC_{old}, so that they can be used by the BCDA algorithm in future cycles to derive information. For example, if no solution could be found to add a 'dependentOn' link from `handle car` to another step, then appropriate rules can be defined for this purpose. These new derivation rules in `handle car`'s default policy alternative can then be used when the BCDA algorithm attempts to deduce this information in step (5) of the BCMA.

ii. Subsequently, the algorithm verifies whether any derivation rules have been removed, that is, if there are any rules 'ders' such that 'ders' $\in$ PA_{old} but not 'ders' $\in$ PA_{new}. For any such rule 'ders' the algorithm checks whether there are instantiations of 'ders' in $DERS_{applied}$. If so, then these instantiations are removed in accordance with the activities in step (4.f.iii.E) of the BCDA. `Garage Inc` can decide for example that nothing should be done after `handle car` has been completed. Therefore it removes all derivation rules leading to this conclusion. If any of these rules were applied already while generating D, then the results must be undone. To this end the algorithm conducts activities conform step (4.f.iii.D) in the BCDA.

iii. Finally, the algorithm determines for all rules 'ders' for which it is true that 'ders' $\in$ PA_{old} and 'ders' $\in$ PA_{new} whether they have been modified. Such modification constitutes the re-definition of a rule's conditions, its conclusion and/or its status (by modifying its activation and expiration date). If such changes are made to a not yet applied rule 'der' (i.e. there are no instantiations of 'ders' that are part of $DERS_{applied}$), then they do not impact the design D. However, if an already applied derivation rule 'ders' was modified, then the algorithm analyzes the impact as follows: first the status of 'ders' is verified. If it turns out that 'ders' was not supposed to be active when it was applied, those instantiations are removed from D conform step (4.f.iii.E) of the BCDA . Next, for all remaining rules 'dsr' the conditions of 'ders', that is $ders_{cond}$, are tested to see if it is still true that $ders_{cond}$ $\in$ D. If not, then 'ders' is no longer true due to the modification and thus ders' instantiation must be removed. The followed procedure in such situations is as described in step (4.d.iii.E) of the BCDA. To exemplify, let us assume that the threshold for when `garage repairer` has to get approval has changed from \$500 to \$400. Then, in a business collaboration where the estimate was \$450 `repair car` would have been initiated, while under the new rule this is not the case. As such, this fact must be removed from D. This same procedure is followed if the status of a rule is affected such that its application has become invalid.

When the conditions of the new 'ders' are still such that $ders_{cond}$ $\in$ D,

the conclusions of the old 'ders' are checked. If there are conclusions in D derived from the old 'ders' but which can not be derived anymore after 'ders' modification, then such conclusions are removed in the manner described in steps (4.f.iii.A) to (4.f.iii.E) in the BCDA. This procedure is also followed when the conclusion of 'ders' has been changed. Also, in case old conclusions are still supported but based on different facts, then the algorithm updates 'ders' instantiations in $\text{DERS}_{applied}$ accordingly. To illustrate, suppose that the earlier mentioned rule of when to initiate repair based on the estimated repair cost changes and `garage repairer` should now get a second opinion first if the estimate is below $500 rather than immediately starting repair. Then, in the existing design D `handle car` must be removed.

iv. *Update control rules*
When all changes have been incorporated, the algorithm next updates the control rules in PLC by replacing the old with the new control rules for all applied policy alternatives. This ensures that during verification of the correctness, consistency and completeness of the design in steps (3) to (5) the algorithm will use the latest control rules.

3. *Check correctness*

When all changes have been incorporated, the algorithm checks the correctness of the design D, that is, whether D is entailed by DCCCR. This is required as the set of correctness rules may have been modified. Any found incorrectness is resolved following the procedure for correctness outlined in step (4.e) of the BCDA. Note that it is possible that problems are found that were not identified prior to the update of the correctness rules.

4. *Check consistency*

Next, the algorithm checks the consistency of the design D, that is, whether D is entailed by DCCNR. Any conflicts that are found, are resolved conform the procedure for consistency checking in step (4.f) of the BCDA. Observe that if the consistency rules have been changed, then inconsistencies may be found in D that were not detected by the old consistency rules in PLC. Also note that inconsistencies found to be unresolvable before, may have become resolvable. Thus the list $\text{DI}_{failed_resolution}$ is re-set prior to the updating of the design schema.

5. *Check completeness*

Once all inconsistencies (if any) in D have been resolved, the algorithm verifies completeness, that is, whether D is entailed by DICMR. Any found incompleteness is subsequently addressed by initiating the BCDA until D is complete. Like in the previous step new incompleteness problems may be found, which had thus far gone unnoticed but are now found by the new and/or modified completeness rules. Observe

that also the list $\mathrm{DI}_{facts_leading_to_inconsistency}$ is cleared, as earlier unresolvable inconstencies may have become resolvable; and thus we should attempt to deduce them rather than exclude them from the list $\mathrm{DI}_{to_complete}$.

This concludes the discussion on the BCMA. In the following two sections we will demonstrate how this algorithm supports dynamism, and undefined adaptability.

6.3.2 Dynamism

Dynamism is the ability to modify business collaboration designs at runtime, and specifically to transform these designs from the old to the new requirements when the underlying design schema is changed. As we observed in the introduction of Chapter 5 the distinction between design time and runtime has a somewhat different meaning in the rule based approach for dynamic business collaboration as design occurs at runtime. Because a business collaboration design is not shaped until runtime, design time changes are considered to be those changes that affect things not carried out yet (and thus have not been designed yet). These changes are identical in terms of timing as those addressed by flexibility and formal adaptability, and they are thus handled in the same fashion as described in sections 6.2.2 and 6.2.2. In contrast, runtime changes are those changes that impact things that are currently happening or have occurred already (and thus have already been designed), and which require a slightly different approach.

There are three options available for handling runtime changes: the first option, abort (and optionally restart), is an option, but as business collaborations typically involve a lot of work, it would be wise in our opinion to avoid the drastic measure of cancellation unless it can absolutely not be avoided. However, it is supported in the rule based approach by stopping the business collaboration, adding new derivation rules and modifying/removing existing derivation rules and subsequently starting again from scratch in accordance with the BCDA. Any changes made to the control rules will also be effectuated when the resulting designs are verified for correctness, consistency and completeness. The second option, doing nothing, is a more feasible one. Here the requirements are changed, but only for new business collaborations. In the rule based approach this is supported via addition of new rules and/or re-definition of existing rules (both derivation and control rules), while making sure that these modifications do not impact running business collaborations. The key to this is the versioning of rules using the appropriate facilities offered by the BCRL (as discussed in section 5.4 of Chapter 5).

To illustrate, let us assume that `Garage Inc` re-defines a control rule to state that "all car repair estimates must be higher than $100" rather than $50, then a new variant of the control rule will be created and applied to constrain new business collaborations. The old control rule, in the meantime, will continue to constraint existing collaborations. In a similar fashion, if the derivation rule in the policy of `estimate repair` changes in the sense that now all repairs above $600 have to be approved, i.e. an existing derivation rule is re-defined. To make sure that existing business collaborations are not affected, this new rule is created as a more recent variant of the existing one. In this manner the 'old' rules

are kept for the running collaborations while the re-defined ones will apply to new business collaborations. In a similar manner, if new rules are added to a policy alternative, then the modified policy alternative does not override the old one but rather co-exists with it as another variant.

To test the new rules the generating of business collaboration designs following these rules can be simulated conform the BCDA. During such simulation any problems can be detected and resolved. Performing of the existing business collaborations, in the meantime, is done conform the old derivation and control rules following the procedure defined in the BCDA algorithm. For example, the above change in the internal process of `garage repairer` will influence the times at which `Garage Inc` will send `car repair report` to `Lee C.S`, i.e. perform task `report estimate`. As such, the protocol of `garage repairer` will need to be changed to reflect this by incorporating a similar rule as the modified one. Otherwise, `Garage Inc` promises to send `car repair report` when estimates are above $500, while this will only be the case when they are above $600. This in turn will result in an incompatibility with the existing operational agreement with `Lee C.S` concerning when `report estimate` will be carried out. `Garage Inc` thus will have to talk with `Lee C.S` to negotiate a re-definition of the agreement. If `Lee C.S` agrees, then the needed changes can be finalized in their respective schemas. In turn `Lee C.S` at this point may need to make modifications to the protocol of `accountant` and subsequently to its internal process because of the change originally introduced by `Garage Inc` (and possibly then to the protocol for the interaction with `AGFIL`). However, none of these changes need to be effectuated in the already running business collaborations between `Garage Inc` and `Lee C.S`.

The third option, that of migration, involves transforming already completed parts of a running business collaboration from the old to the new situation when the underlying design changes. In context of the rule based approach such transformation is necessary in two situations. The first is when already applied derivation rules are changed and these changes must be reflected in the design. In such event the procedure outlined in step (2) of the BCMA is followed. To exemplify this, let us assume that the derivation rules for `garage repairer` concerning when to ask for approval have changed. We assume that the threshold for when to get approval and when to initiate repair has changed from $500 to $400 in their conditions. Specifically, informally the two modified rules are "if repair cost below $400, then do repair car" and "if repair cost above $400, then get approval". We also suppose that we have a business collaboration where the estimate was $450. Consequently, on the basis of the old rules `repair car` was initiated, and it is at this point that the change is taking place. Note that if communication with `Lee C.S` had already occurred, then this would have had to have been undone as well for the reasons described in the previous paragraph.

Following the defined procedure we check the effects of changing the conditions of the two derivation rules. Since the estimate is above $400, this means that the conditions of the first derivation rule are no longer met by the design. As such, the addition of a 'dependentOn' link to `repair car` is invalid. Then, conform step (2.b.iii) of the BCMA the task `repair car` is removed from the design (as well as any related information as

described in step (4.f.iii.A) of the BCDA). Also, assuming that in `repair car`'s policy a rule has been specified that if it is removed from a design the task `stop repair` must be added, this rule is applied (in accordance with step (4.a) in the BCDA). As such, this compensating task will be added to the design. The modification of the second, as of yet unapplied, derivation rule is resolved via the activities described in (2.b.iii) of the BCMA, i.e. it is included in the set of available rules. Then, since the 'dependentOn' link was removed from the design, this incompleteness is detected in (5) of the BCMA. Consequently, in the new cycle of the BCDA the algorithm will look for solutions to derive this link again. The algorithm will then find a solution using the unapplied modified rule. The conditions of this rule were not true in the design D before, but they are true now. As such, the derivation rule is applied leading to addition of a 'dependentOn' link to the `get approval` task. Thus, as a result of the changes `repair car` will be cancelled by `stop repair`, whereas at the same time the new correct path via `get approval` will be followed at runtime.

The second situation requiring transformation of a design is when its control rules change. Recalling the correctness rule "all car repair estimates must be higher than $50" of `Garage Inc`, let us suppose that this rule is modified to "all car repair estimates must be higher than $100". Postulating that in an existing business collaboration an estimate was made of value $75, this means that such estimate was invalid and must be undone (and all subsequent actions). The effectuation of this change is straightforward. Assuming that the derivation rules have not changed, the BCMA passes through most of its first two steps unhindered. However, conform step (2.c) the BCMA then updates the applicable control rules including the just modified correctness rule. Then, in step (4) the correctness of the business collaboration is evaluated resulting in the detection of the incorrectness. Consequently, the made estimate will be removed from the collaboration design. Also, any actions that have been taken already on the basis of this estimate will be undone. Let us suppose that the modified control rule is incorporated while `Garage Inc` is conducting `repair car` (conform its derivation rules as mentioned above). After that, conform the procedure described in the BCDA the 'dependentOn' link to this task will be removed as well from the design. Following the earlier made assumption that in `repair car`'s policy a rule has been specified that if the link is removed from the design the task `stop repair` must be added, this rule is applied (performing step (4.a) in the BCDA). As a result the compensating task will be added to the design. The design is next then checked for consistency in step (4) of the BCMA and subsequently for completeness in step 4(5), e.g. leading to make another estimate of the `car repair cost`.

6.3.3 Undefined Adaptability

Lastly, undefined adaptability expressed the ability to handle unforeseen changes that occur during the carrying out of business collaborations. Many of these changes will be too severe to be handled by the system responsible for business collaboration development and management itself (such as power failures); and these are thus not taken into consideration in this dissertation. The handling of less severe unexpected events that occur though is

supported by the BCDA and BCMA. The procedure in this regard is largely the same as discussed for handling dynamism: when an undefined event takes place, this will become apparent by the fact that the BCDA can not find any derivation rules that define how to handle it. As such, the algorithm ends up asking the user for assistance. At this point the user can modify the rules in such a way as to deal with the event, which are then incorporated in the design following the BCMA. This can entail the addition of new derivation rules conform step (2.b.i) in the BCMA, but also the removal and/or modification of existing derivation rules following steps (2.b.ii) and (2.b.iii) in the BCMA. These changes are then incorporated in the design after which the unexpected event can be handled. Note that alternatively it is possible that appropriate derivation rules were defined already, but their application is prohibited by the current control rules. Modification of the control rules is another option then in addition to the (re-)definition of the derivation rules.

To demonstrate, let us assume that `Garage Inc`'s service `car repair service` sends the message `repair estimate request` to `claim management service` of `Lee C.S`. Let us also suppose that rather than receiving the standard confirmation response, `car repair service` receives an unexpected timeout error message. As there have been no rules specified that depict how to handle such message, the BCDA will be unable to deduce how the business collaboration is to proceed. Therefore, the responsible rule steward is contacted to resolve the problem. The rule steward of `car repair service` will then define new rules that will address such error message receipt. Next, the steward will contact `Lee C.S` to negotiate about the proposed changes to the agreement (governing under which conditions such error message will be sent). If approved, the agreement will be adjusted. After that, the steward assesses the impact of this change on the protocol of `car repair service` and consequently on its internal workings. After that the error can be handled correspondingly by `Garage Inc` and execution can continue. In this process `Garage Inc` and `Lee C.S` can evaluate different modifications to the agreement by simulating the consequences of such changes to the protocols and processes of their respective services.

Alternatively, let us hypothesize that such derivation rule had indeed been defined already stating basically that the message is to be re-send, i.e. operation `send estimate` has to be performed again. However, this is not allowed by the consistency rule that `Garage Inc` currently has associated with this operation in the protocol of `car repair service` informally saying that "if operation `send estimate`, then it is not allowed for the operation to be dependent on itself". Such rule prevents the looping behavior as is mandated by the agreed upon derivation rule. Thus, if `Garage Inc` modifies the `car repair service` protocol no inconsistency will be detected by the BCMA, as the consistency rules are updated in step (2.c). The desired conclusion can then be deduced in step (5) of the algorithm when completing the design. This shows that the procedure to deal with undefined adaptability is almost the same as that for dynamism. The difference with dynamism lies herein that it is known what dynamism change can occur. Consequently, derivation and control rules to cope with these changes can be pre-defined. In contrast, with changes in the undefined adaptability category such rules can only be specified at the moment the change occurs, since this change was unknown up till that point in time.

6.4 Discussion

In this chapter we built on the preliminaries developed in Chapter 5 to develop the rule based approach for business collaboration outlined in section 5.1. The idea of using rules to make the automation of business collaborations more dynamic is not new. Several representative rule based approaches found in literature are (Casati et al., 2000), (Medjahed et al., 2003), (Paschke, 2005), (Shankar et al., 2002) and (Zeng et al., 2003). (Casati et al., 2000) uses rules to facilitate dynamic service selection in their eFlow system, so that services can be selected at runtime when workflow tasks need to be performed. (Medjahed et al., 2003) uses a rule based approach to create composite services from high level declarative descriptions. The method uses composability rules to determine whether two services are composable. Multiple plans may be generated in which case the user can select the most desired one for example based on cost, quality of service, and security. A work resembling the latter is SWORD (Shankar et al., 2002) in which the service requester defines the begin and end state desired after which a plan is generated to move from the first state to the next using a rule-based expert system. (Zeng et al., 2003) describes the rule inference framework DYflow in which rules are used to drive the development of service compositions. Depending on the specific requirements services are composed on the fly, where decisions concerning their ordering and binding are governed by rules. The difference with (Medjahed et al., 2003) and (Shankar et al., 2002) is that in (Zeng et al., 2003) compositions can be altered at runtime by rule modifications, whereas in the other works a composition is generated at design time and consequently carried out.

Compared to the mentioned approaches the presented work differs in several ways. With regard to the development and management of rules, rule based approaches like the ones described in (Casati et al., 2000), (Shankar et al., 2002) and (Zeng et al., 2003) assume that rules are simply defined in a correct and consistent manner. However, when we consider the magnitude, diversity and complexity of the rules that organizations follow, the invalidity of this assumption quickly becomes apparent. Without proper mechanisms to help them developers will be at a loss when trying to ensure that developed rules and policies are and remain complete, correct and consistent. This problem has been recognized in many works dealing with rule verification (e.g. (Aiken et al., 1995), (Bailey et al., 2002), (Baralis et al., 1998), (Leemans et al., 2002) and (Wu, 1993)), and the mechanisms presented in section 6.1.2 are largely based on these works. In several ways though the mechanisms developed in this section for rule verification and validation are more extensive. For example, we deal with additional issues such as non-monoticity and prioritization, versioning and status when checking for ambivalence, circularity and redundancy. With regard to deficiency we use a relatively simple approach utilizing completeness rules rather than develop a generic detection mechanism. However, as we argued for in section 6.1.4 such mechanism suffices for our purposes.

When it comes the application of rules, the rules and policies that we employ are much richer in nature and scope. For example, the Business Collaboration Design Algorithm takes the possible non-monoticity and prioritization of rules into consideration, something which is not done in the mentioned related works. By comparison, (Zeng et al., 2003)'s

rules do not exceed the complexity of simple Event-Condition-Action (ECA) rules, whereas (Shankar et al., 2002) uses basic Horn rules. (Grosof, 2004) does consider these matters, but focuses on specification rather than application. Also, the Business Collaboration Design Algorithm can deal with modalities in rules, and distinguish between rules with different status and version when applying rules. Moreover, the types and scope of rules greatly extends that of existing solutions. (Zeng et al., 2003) for example only considers control flow, data flow and resource selection type of rules. The approach in (Shankar et al., 2002) is more generic, however, only focuses on rules in relation to service composition. In contrast, in the BCDA a wide variety of derivation rules is applied during design at runtime; in particular when considering that we do not limit ourselves to service composition rules but also include higher level business requirements (as well as both private and public processes). Additionally, the BCDA takes multiple policy alternatives into consideration and determines which is best to apply in what situation. As such, the approach offers much more dynamicity for business collaboration in terms of which part of a collaboration can be dynamically designed and subsequently carried out in comparison to other proposals.

Additionally, in our approach the order in which incompleteness problems are resolved is not fixed. As such, the order may be set in any manner to support a preferred style of design. This distinguishes the approach from works like (Shankar et al., 2002) and (Zeng et al., 2003), which adopt forward chaining in the inferencing process. The flow based rule processing underlying the BCDA has the advantage that automatic rule chaining and re-evaluation is prevented. Also, not all possible flow paths have to be checked, since the logical flow is based on the defined order in which rules are applied. This makes the algorithm more efficient compared to other works based on generic forward chaining. Another area in which the presented approach compares positively to other works is related to the maintaining of validity, alignment and compatibility of business collaboration designs. In the Business Collaboration Design Algorithm each incremental development of a design is checked using control rules. (Shankar et al., 2002) and (Zeng et al., 2003) do not mention such issues at all. (Casati et al., 2000) does consider them but only with regards to verification of consistency using so-called consistency rules. Interdependencies between models at different levels and/or aspects are not taken into account. (Medjahed et al., 2003) employs a similar notion in the form of composability rules to compose semantical services, but it suffers from the same limitations.

Concerning the management of existing business collaboration designs, rule approaches such as in (Casati et al., 2000) and (Zeng et al., 2003) facilitate support for flexibility and formal adaptability in the sense that the employed forward chaining application of rules allows for the definition of different rules for handling different situations. However, both lack the structured manner in which the Business Collaboration Management Algorithm supports such dynamicity in the form of policies and policy alternatives. With regard to dynamism and undefined adaptability no support is offered by these solutions. (Shankar et al., 2002) has the means for supporting dynamism and undefined adaptability due to its capacity to perform non-monotonic reasoning. However, the different types of rules that are considered, are more limited than as presented in this chapter. As such, the types of changes that can be easily accommodated are much more limited. In addition, modalities

of rules are not taken into account. Furthermore, whereas in the BCDA and the BCMA the correctness, consistency and completeness of generated and managed business collaboration designs is ensured, this is not the case in (Shankar et al., 2002). (Medjahed et al., 2003) offers some support via its composability rules but only in relation to the consistency between protocols. This is an essential issue as lack of support for such verification means that the impact of changes to business collaborations can not be properly assessed; something that is particularly required when changes affect other parts of the business collaboration designs such as changing technical requirements leading to shifting business requirements and vice versa.

The same advantages of the presented approach can be found when compared to other (mostly non-rule based) works on dynamicity discussed in section 2.3 of chapter 2. When it comes to flexibility, solutions like (Sadiq and Orlowska, 2000), (Georgakopoulos et al., 1995) and (Curbera et al., 2002) offer limited support. Typically they focus on control flow modification and/or dynamic resource binding, whereas we take a much wider variety of changes into consideration. With regard to formal adaptability most solutions employ pre-defined exception handlers, such as done in (Eder and Liebhart, 1996), (Curbera et al., 2002) and (Hagen and Alonso, 2000). This has the problem that it is impossible as well as undesirable to pre-define all possible exceptions. Moreover, including many exceptions makes models very complex. In the approach that we propose rules can be defined to handle exceptions just like normal changes. Moreover, such rules can be added at runtime if an unexpected event occurs. In this sense the approach is similar to the one in (Brambilla et al., 2005), which uses a combination of exception handling and user based intervention. The usage of rules for formal adaptability support is akin to for example (Li et al., 2003), where the modeling and handling of exceptions relies on continuations, listeners as exception handlers, and on policies, or strategies, for continuation. In relation to dynamism works like (Georgakopoulos et al., 2000) and (Reichert and Dadam, 1998) have aimed at supporting modifications to running processes. However, the types of modification that are allowed is more limited when compared to the approach described in this chapter. The usage of control rules to constrain modifications in the presented approach is similar to ideas proposed in (Casati and Shan, 2001), (Geppert and Tombros, 1998) and (Meng et al., 2002), but we take this notion beyond just changes to the control flow structure of business collaborations. Lastly, undefined adaptability is not really considered at all in the above mentioned works, whereas we showed that in our approach such changes can be facilitated as well by simply adding appropriate rules at runtime (excluding severe errors like a complete system failure or power outage).

Now, a concern that may be raised in relation to the dynamicity provided by the proposed rule based approach, is that it is complex in nature involving the definition of a large number of BCIM elements and their policies necessary to generate the required models and mappings; which moreover need to be finely tuned to generate consistent business collaboration designs. One argument against this concern is that by nature business collaborations tend to be complex in nature and as such any approach that faithfully wishes to tackle the problem of their development and management will unavoidably be complex as well. With regard to the provided support for dynamicity we argue that in the type of

business collaboration considered in this dissertation such support is necessary due to their ever changing nature. In combination with the developed mechanisms to control business collaboration design we feel that our approach is then more suited for supporting the different forms of dynamicity than other developed solutions; as these mechanisms reduce the effort required for the modification of business collaborations.

For less dynamic business collaborations it may be argued that the offered dynamicity is not required. This is a valid point and this was also the reason that such collaborations were excluded from the research scope in section 1.5 of Chapter 1. We wish to note though that the design of these more static collaborations can also be accommodated. Organizations can develop static business collaboration designs using the model based approach in Chapter 4, where the requirements will be expressed as rules without conditions (i.e. be always true) in the context of the rule based approach. Then, the generating of the design can still be done conform the BCDA (albeit that this will always result in the same design). Moreover, this allows the consistency of these requirements to be still verifiable using the pre-defined domain independent control rules (plus any additionally defined control rules). If at some point it turns out that changes do have to be made, the BCMA can then be simply followed to incorporate them into the design. In this sense static business collaborations are a special case in the proposed approach in which all derivation rules in the design schema constitute facts; and thus all interpretations of the design schema result in the same design.

Concluding, we can state that with regard to the fifth and sixth research question we defined in section 1.6 of Chapter 1 this chapter has provided us with the following answers:

1. We first introduced mechanisms that aid organizations with the development of design schemas. Concretely, we created four detection mechanisms for the analysis of the rules and policies of BCIM elements in these schemas with regard to ambivalence, circularity, deficiency and redundancy anomalies. This provides organizations with the means to develop schemas for their business collaborations that are complete, correct and consistent; and have clear and unambiguous semantics.

2. We then resolved the issue of how to administer the rules in design schemas in order to generate designs. We started by developing definitions for verification of conformance and validity. Based upon well established ideas concerning Datalog semantics these definitions enable us to verify that business collaboration designs are conform requirements and moreover that these designs are complete, correct and consistent. After that we defined a generic Business Collaboration Design Algorithm grounded on these mechanisms with which designs can be generated through rule application. We subsequently showed how this algorithm supports both flexibility and formal adaptability.

3. We next explored the impact of changes to a design schema to already running business collaborations. Concretely, we developed a Business Collaboration Management Algorithm and explained how, based on the Business Collaboration Design

Algorithm, it supports the runtime adaptation of designs if their underlying schema changes; as such accommodating both dynamism and undefined adaptability.

Based on the above we can conclude that we have successfully acquired the answers that we sought throughout this chapter. This has resulted in an extensive and cohesive rule based approach with which dynamic business collaboration development and management becomes feasible. In the next part of this dissertation we will discuss the implementation of the proposed approach through the development of a prototype to demonstrate its practical feasibility.

Chapter 7

Prototype

Any sufficiently advanced technology is indistinguishable from magic; Arthur C. Clarke

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning; Rich Cook

In Chapter 5 and 6 we introduced and explained a rule based approach for dynamic business collaboration development and management. Concretely, we developed mechanisms for the classification, specification, application, and management of rules in the context of business collaboration that built on the ideas developed in Chapter 3 and 4. Throughout these chapters we applied the approach in the AGFIL case study to demonstrate its usability. We also provided formal underpinnings to showcase the approach's logical consistency. In this chapter we complete the validation of the presented research by providing proof of the implementability of the rule based approach. In terms of the research road map discussed in section 1.3 of Chapter 1 this constitutes the last research step (as illustrated in Fig. 7.1).

As the figure shows the result of the final road map step leads to the creation of a conceptual architecture and corresponding prototype for a system capable of developing and managing rule based business collaborations. We call such system a *Rule Based Business Collaboration System (RBCS)*. This system must implement the approach as described in Chapter 5 and 6. Concretely this means that an RBCS must facilitate the following:

1. The specification of design schemas for business collaborations, that is, of which BCIM elements a business collaboration design can consist and the policies associated with these elements.

2. The analysis of developed (or modified) design schemas to verify that the defined BCIM elements and policies do not contain any anomalies as described in section 6.1.

3. The application of the policies of the BCIM elements to drive and constrain the

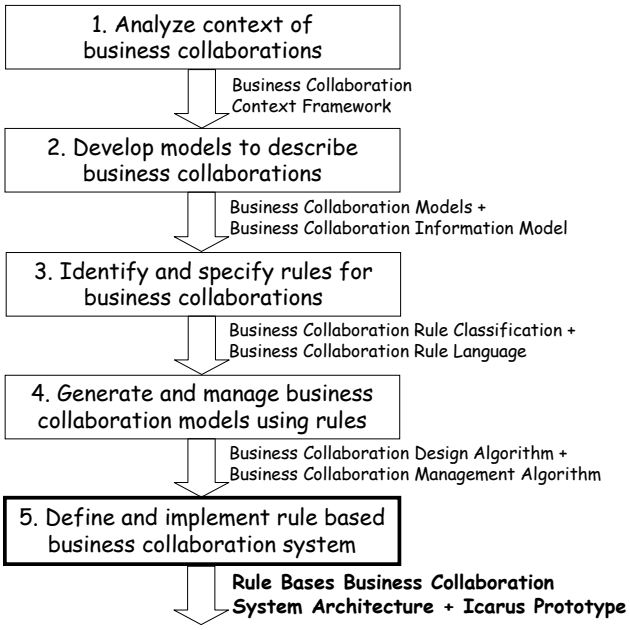

Figure 7.1: Research Road Map - Implementing A Rule Based Business Collaboration System

generating of business collaboration designs. This requires the RBCS to implement the Business Collaboration Design Algorithm (BCDA) described in section 6.2.2 of Chapter 6.

4. The runtime generating and carrying out of these business collaborations designs based on the specific circumstances of individual collaborations and the policies and rules in their associated design schemas.

5. The management of the design schemas of business collaborations and the impact of changes on existing corresponding designs. This means that the RBCS must implement the *Business Collaboration Management Agorithm (BCMA)* algorithm as defined in section 6.3 of Chapter 6.

Now, a key issue that needs to be addressed when creating an RBCS capable of the above, deals with how rules are to be integrated with business collaborations. In line with the discussion in section 1.4 of Chapter 1 questions that must be answered include: are rules an integrated part of business collaborations or are they separated into components/services? Should the policies and rules in business collaboration design schemas be centrally managed or should they be decentralized within organizations? How can we design a rule engine to drive and govern business collaboration development and management using the specified policies and rules in the design schemas in accordance with the *BCDA* and *BCMA*? In the remainder of this chapter we provides answers to these questions. For

this purpose the chapter is structured as follows: first in section 7.1 we analyze the possible ways in which rules can be integrated with business collaborations, and argue the advantages of adopting a service oriented rule integration approach by using rule engine technology. We also discuss how such rule engine can offer support for the development and management algorithms (i.e. the *BCDA* and *BCMA* algorithm respectively) as described in Chapter 6. After that we introduce the Icarus prototype and we explain how Icarus provides an implementation of a rule based business collaboration system centered around a sophisticated rule engine; as such facilitating the development and management of business collaboration design schemas and designs. Finally, in section 7.3 we use examples from the AGFIL case study in a laboratory experiment to illustrative the workings of the prototype for developing and managing business collaboration designs.

7.1 Integrating Rules And Business Collaborations

As observed in the introduction of this chapter the integration of rules with business collaborations is a key issue, which has to be resolved in order to implement a rule based business collaboration system. (Rosenberg and Dustdar, 2005) provides us with a starting point in this regard, identifying three options such integration: code-based, model driven, and service-oriented integration. Code-based implementation constitutes hard-wiring rules into business collaboration applications. This is obviously not what we want, since the whole purpose of the rule based approach is to make design more dynamic, not less. Scripting and rule components can help fix this problem to some extent, however rules remain difficult to manage in those cases as well. In model-driven incorporation the rules are separated from actual designs and can be changed independently. Rules can be easily specified and managed as they share the same context as the modeled business collaborations that they constrain. Rules are defined, administered and managed using a separate specialized component external to the process, typically a *rule engine*, where the rule engine is tightly integrated with the actual design tool.

By and large this seems like the approach that we have been advocating throughout this chapter. (Rosenberg and Dustdar, 2005) notes though that model-driven incorporation has an important disadvantage which comes from the notion that often in a business collaboration the rules will require context additional to the one that can be provided by this collaboration. For example, assessment of a claim by `AGFIL` may be based on data not directly related to the particular claim such as the number of claims already granted this month. As this information is not contained within the design, the rules can not refer to it. Thus the information is not available to base decisions on. Therefore, (Rosenberg and Dustdar, 2005) proposes to adopt a service oriented integration approach. In such approach rules are kept in a centralized location, which is accessible by all business collaborations throughout the organization. Rules are deployed to the rule engine, and are then exposed via a decision service. By keeping the rule engine up-to-date with facts as they occur, at any point in time that a business collaboration needs to make a decision it can invoke the decision service; where the outcome will be based on all known relevant facts.

Although we also argue the necessity for a service oriented approach to integration in this dissertation, we feel that the reasoning in (Rosenberg and Dustdar, 2005) is conceptually flawed because the retrieval of information is in our view part of the activities performed in a business collaboration. For example, the retrieval of the number of granted claims over the last month may be part of `AGFIL`'s claim handling process. In the approach advocated by (Rosenberg and Dustdar, 2005) such information would be added to the rule engine by external sources. As such this information flow is only implicitly present in a business collaboration. This makes the business collaboration less manageable in situations where changes affect this activity. This does not mean that we do not advocate usage of service oriented integration, however, our reason for adopting such integration is founded on a different rationale. A service oriented approach offers the most flexible integration of the rule engine and the engine executing the business collaborations due to the loosely coupled nature of services. By providing the functionalities of the rule engine as a service multiple execution engines can use the same rule engine (possibly providing different sets of pre-defined completeness rules to the rule engine in order to vary the manner in which designs are generated). A service oriented integration of the rule engine also makes it easier to replace the rule engine with a different implementation. At the same time it enables rules to be reused and applied across business collaborations as well as used by other applications.

In the remainder of this section we will explore the implementation of rules via a service-oriented integration approach in more detail. Specifically, we first analyze the notions of rule engines and rule inferencing in section 7.1.1. Then, in section 7.1.2 we explore how a rule engine can be used to implement the Business Collaboration Design Algorithm and the Business Collaboration Management Mechanism discussed in section 6.2.2 and in section 6.3 of Chapter 6 respectively.

7.1.1 Rule Engines And Rule Inferencing

In the introduction of this section we argued in favor of using a service-oriented approach to incorporate rules in business collaboration designs. In this argument we alluded already to the usage of rule engines as the entities responsible for the definition, application and management of rules. Rule engines are software applications that contain definitions of rules (Chisholm, 2004), and they are the typical mechanism with which rule specification languages like the ones in section 5.4 are implemented (see (d'Hondt, 2005), (Rosenberg and Dustdar, 2005) and (Russell and Norvig, 2003)). Referred to by (Ross, 2003) as business logic servers, rule engines control the selection and activation of rules. Sometimes also referred to as an inference engine, a rule engine activates a rule when incoming data matches either its conditions or conclusions. As such, the engine is responsible for taking declarative statements, and automatically ordering, merging and applying them.

Rule engines are typically classified into production systems and logic systems (d'Hondt, 2005). Production systems are usually data-oriented and are primarily used to manage information. In contrast to these production systems, logic-based systems utilize logic programming for problem solving by inference, e.g. to answer question, infer new knowledge,

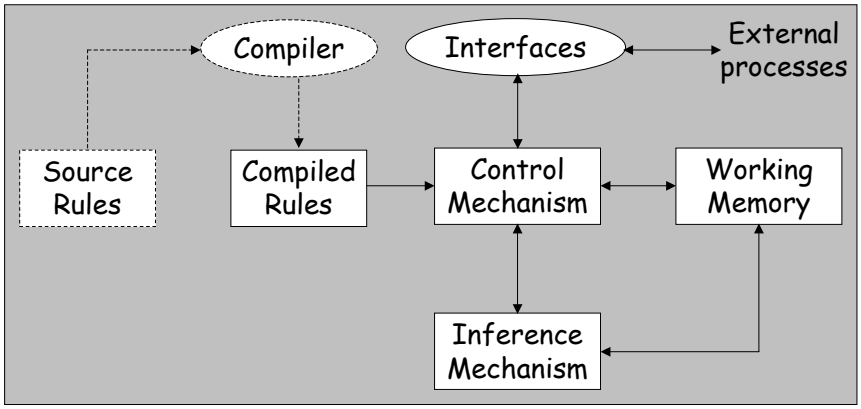

Figure 7.2: Main Elements Of A Rule Engine

and so on. Expert systems are an example exponent of these kinds of system as well as Prolog (Flach, 1994). For an extensive survey of these and other systems (d'Hondt, 2005) provides the reader with a good starting point. To a large extent the make-up of a rule engine is the same whether it is a production or logic-based system. (Chisholm, 2004) provides interested readers with a detailed instruction on how to build a rule engine. Here we only briefly discuss the different rule engine components and their function (illustrated in Fig. 7.2).

The figure shows the components that can be commonly found in a rule engine, albeit sometimes under different names. The 'source rules 'are the statements resulting from expressing rules in an appropriate language. Source rules are typically defined off-line and compiled into an efficient internal representation, though it is possible that they are directly interpreted. Once compiled by the 'compiler' the rules are saved in a private data structure. This makes them accessible to the 'inference mechanism' whose function it is to apply rules at runtime. Application of a rule leads to a change in the 'working memory'. The working memory is a private data area which keeps track of the state of the data affected by the rules.

A key role in this process is reserved for the 'control mechanism'. The control mechanism is responsible for coordinating the actions of the other components in the rule engine. Firstly, the mechanism functions as a liaison between internal activity and the outside world by allowing 'external processes' to interact with the rule engine via its 'interfaces', more specifically to update the working memory with new facts or remove existing ones. Secondly, the control mechanism determines what rules are applied and in which order.

Rule application, also referred to as rule firing, transpires using an agenda. Based on the current state of the working memory the set of useful rules is extracted from the compiled rules, and these are put on the agenda. Then, in accordance with a ranking scheme the rules are ordered and the highest-ranking rule is applied. Due to rule application the facts in the working memory may then change resulting in an update of the agenda. This continues until all rules have been applied, or until the application of rules no longer leads to the modification of the working memory. This process is referred to as *inferencing*.

In general there are three forms of rule inferencing possible, being *forward chaining*, *backward chaining* and *flow based rule processing*. In forward chaining the inference mechanism starts with the facts available in the working memory and attempts to derive new facts through inferencing. For each rule on the agenda (determined by the control mechanism) the inference mechanism tries to proof its conditions in the if clause, and when successful asserts the its conclusion in the then clause. Note by the way that the control mechanism will only put rules on the agenda whose conditions can be proven. As such inferencing, once commenced, will always succeed. From a semantic point of view forward chaining is often associated with data driven inferencing (or forward reasoning). Design, scheduling and assignment tend to be data-driven in nature, as such employing forward reasoning (Schreiber et al., 1999).

In contrast, in backward chaining the inference mechanism starts with one or more goals that need to be proven. Essentially it searches the available rules until it finds one whose then clause matches the desired goal. It then attempts to prove the if clause of this rule. If unsuccessful, the if clause itself is added to the list of goals as a sub goal. This cycle continues until all (sub) goals have been proven, or the impossibility of doing so has been established. Backward chaining is linked to goal driven inferencing (or backward reasoning). Classification, diagnosis and assessment are typical goal-oriented activities. It is pointed out though in among others (d'Hondt, 2005) that reasoning is not necessarily the same as chaining. As such, it is quite possible to express both types of reasoning in terms of either chaining mode, although the natural combinations result in more expressive results. Both forward and backward chaining are typically implemented using the RETE algorithm (Forgy, 1982).

Alternatively, rule application can follow a pre-defined order of rule firing, which is typically referred to as flow based rule processing. In flow based rule processing a single rule is fired, the working memory is updated, and based on the new state of the memory another rule is selected and fired. The difference between the two is that cyclic firing the assumption is made that rules within a cycle are executed as if the working memory does not change when rules are applied. This allows rules to be specified in a declarative manner. In contrast, in flow-based processing rules are applied in sequence. As such the order of firing becomes important as the working memory and agenda are updated in between the application of rules. Thus, a different order can lead to a different working memory state. In the following section we will explain how we can use a rule engine to implement the generating and managing of business collaborations using these different forms of inferencing.

7.1.2 Generating And Managing Business Collaborations

In the previous section we briefly sketched the different components of a rule engine and described their purpose. In relation to the generating and managing of business collaborations these components informally speaking work together as follows: the source rules of a rule engine are formed by the policies of the BCIM elements in the business collaboration schema expressed in the BCRL business language. These are then compiled into a private structure that is grounded on the BCRL executable language. When a business collaboration design needs to be extended, the control mechanism takes the compiled rules and compares the available derivation rules to the working memory in order to find solutions. The content of the working memory is the set of modeling description atoms constituting the business collaboration design under development. The control mechanism orders the found solutions found into a rule agenda. The solutions on the agenda are next applied by the control mechanism via the inference mechanism through flow based rule processing. As a result the working memory is updated, i.e. the business collaboration design, is updated and returned. If at some point the schema associated with a business collaboration change, then this means that the source rules of the rule engine will be updated. The control mechanism will then assess the effects of the made changes on its working memory and modify the working memory accordingly.

Concretely, the generating and managing of business collaboration designs conform the Business Collaboration Design Algorithm and Business Collaboration Management Algorithm (as discussed in section 6.2.2 and 6.3 of Chapter 6 respectively) is implemented in a rule engine as follows :

1. **Check completeness**

   Because of a request for information needed on how to proceed with the business collaboration (e.g. due to an incoming message) the rule engine starts by performing a completeness check. The rule engine takes the current design as its input and attempts to proof all relevant completeness rules through backward reasoning. Any found violated completeness rules are then ordered by priority.

2. **Find solution(s)**

   For all violated completeness rules the rule engine then attempts to extend the design such that these rules are no longer violated. For this purpose the rule engine starts to look for solutions that lead to the desired extension of a business collaboration design. This is done as follows:

   (a) *Determine applicable derivation rules*

   The control mechanism first determines which of the compiled derivation rules are suitable for application to make them accessible for the inferencing mechanism. Inactive rules and rules from disallowed policy alternatives are excluded at this point.

(b) *Find potential solutions*

The control mechanism then attempts to find solutions for the requested extension through backward chaining. Based on the current design (represented as a collection of facts in working memory) and the set of applicable derivation rules present in the compiled rules the rule engine attempts to proof the goal (i.e. the to-be-derived fact). As a result of the backward reasoning the rule engine produces a list of possible solutions.

(c) *Assess potential solutions*

The control mechanism next checks the found solutions for feasibility. The remaining set of solutions then form the basic agenda of the control mechanism.

3. **Order solution(s)**

Next, the control mechanism sorts the solutions on the agenda. The result of this ordering process is that the rule agenda will contain solutions in order from highest to lowest rank. Note that solutions can contain multiple rules, which is different from normal rule engines where the agenda consists of individual rules.

4. **Apply solution**

Once the control mechanism has been determined, it next starts to apply solution(s). The affect of a successful application of a solution is a transformation of the working memory from one valid state to another. If the new working memory state is not valid (that is, incorrect or inconsistent) or if not all the rules in the solution could be applied, then the rule engine will revert back to the old working memory state. Subsequently, the next solution on the agenda is tried. This is different from a normal rule engine employing forward chaining to derive new facts, as in such process all rules on the agenda will certainly be applied. The application of a solution is done by the control mechanism by applying each of the solution's derivation rules in a flow based rule processing manner using the inference mechanism. This is done as follows:

(a) *Instantiate alternative guards*

To start with the control mechanism determines whether the to-be-applied rule is part of an alternative on its list of already applied alternatives. If not, then the control mechanism attempts to instantiate the guard conditions of the alternative via the inference mechanism. If the inference mechanism does not find any instantiations of the guard conditions, then the control mechanism does not try to apply the rule. Otherwise, the control mechanism puts the found guard condition instantiations on a list of policy alternative guard instantiations.

(b) *Instantiate derivation rule*

Assuming that the guard conditions alternative to which the to-be-applied derivation rule belongs were met, the control mechanism then provides the inference mechanism with this rule. The inference mechanism then deduces the

rule's instantiations based on the current working memory and returns them to the control mechanism. The control mechanism then attempts to add each resulting conclusion as follows:

(c) *Check conclusion*

If the conclusion(s) constitute links or attributions, the control mechanism first verifies if the policies of both BCIM elements in the link/attribution support it. For this purpose the control mechanism attempts to proof the link/attribution using backward reasoning. If the control mechanism can not find two distinct solutions in which the derivation rules of either element are used, then it will not add the fact to the working memory.

(d) *Add conclusion*

If the result of the check was successful (or if the new fact does not constitute a link or attribution), the inference mechanism subsequently adds the fact to the working memory. The control mechanism then stores the rule instantiation in a list that keeps track of which instantiations led to which fact.

(e) *Check correctness*

After that the control mechanism uses backward reasoning to check the correctness of the new working memory state. To this end the control mechanism determines the set of to-be-used correctness rules and attempts to find violations to them based on the working memory. If the control mechanism finds any violations, then this means that the working memory state is incorrect. Consequently, the control mechanism reverts the working memory back to its old state and updates the list of rule instantiations. If the to-be-added fact was crucial for the solution as a whole, the control mechanism discards the current solution and selects the next one on the rule agenda. Otherwise, it continues with applying rules from the current solution via the inference mechanism.

(f) *Check consistency*

If the new working memory state was found to be correct, the control mechanism again uses backward reasoning, but now to test for consistency using the applicable consistency rules. The result of the inferencing is a list of violated consistency rules. If none were found, the control mechanism continues to check completeness. Otherwise, the control mechanism attempts to resolve the found inconsistencies as follows:

i. Identify conflicting rules
For each violated consistency rule the control mechanism identifies which facts in the working memory instantiated the consistency rule leading to this fact. Subsequently, the control mechanism determines what derivation rules were responsible for each of these facts by using its list of instantiations.

ii. Analyze priorities
For each of the found derivation rules the control mechanism then examines the ranking and prioritization information, and compares them to determine

which derivation rule has least preference. Its conclusion is then selected as the to-be-removed fact.

iii. Resolve inconsistency
If this is the just-added fact, then the control mechanism reverts the working memory back to the old state. The control mechanism also updates the list of rule instantiations. If the to-be-added fact was crucial for the solution as a whole, the control mechanism discards the current solution and selects the next one on the rule agenda. If an already applied derivation rule is to be undone, then the control mechanism conducts the following activities:

A. Firstly, the control mechanism examines if the removal of the fact jeopardizes the currently being applied solution. If so, it reverts the working memory back to the old state and updates the list of rule instantiations. Otherwise, the control mechanism removes the fact from the working memory. Simultaneously it adds a fact conveying this removal.

B. Secondly, the control mechanism identifies if there are any derivation rules that have become instantiated due to the removal of the fact. Specifically, the control mechanism examines if there are any source rules whose conditions are (partially) based on such removal. If so, then the control mechanism takes those derivation rules' conclusions and attempts to proof them through backward reasoning. If any solutions are found, the control mechanism consequently applies them in the manner as described thus far in (3).

C. Thirdly, the control mechanism verifies the rule instantiations made thus far to see if any of them have become invalidated due to the fact removal. If an alternative's guards are no longer supported by the working memory, then the control mechanism will undo all of the rules applied from this alternative. Next, the control mechanism checks each individual rule instantiation to see if it is still supported by the working memory.

D. Fourthly, the control mechanism removes all violated instantiations of non-monotonic derivation rules. For monotonic rule instantiations it contacts the user, where user contact is mediated via the rule engine's interfaces. If all instantiations underlying a fact are removed, then the control mechanism removes this fact from the working memory.

E. Fifthly, when all violations have been handled, the control mechanism checks the consistency of the working memory again through forward reasoning based on the applicable consistency rules in the compiled rules. If any inconsistencies are found, then the control mechanism goes through the steps in (3.d) again. Otherwise, the control mechanism goes to (4).

(g) *Update rules*
To prevent usage of derivation rules from policy alternatives no longer available for reasoning because another alternative has been chosen already, the control

mechanism lastly updates its list of applied alternatives. This also has the affect that only the relevant control rules will be taken into consideration in future design.

5. **Check completeness**

   After the control mechanism has applied all the solutions on the rule agenda successfully, it concludes by performing another completeness check. This check is once again carried out through backward reasoning, now with the set of completeness rules associated with the BCIM elements in the new working memory. If the resulting list of violated completeness rules is empty, then the control mechanism stops reasoning. Otherwise, the control mechanism evaluates each violated completeness rule, determines what fact is missing and tries to derive it following the just described procedure.

6. **Change rules**

   In case the design schema(s) associated with the business collaboration change and these changes must be effectuated to the design under development, then the source rules of the rule engine will be updated accordingly. Subsequently, the control mechanism will analyze the new source rules and determine if and how they affect the current working memory in the manner as described in the BCMA. As the steps in the BCMA build on those in the BCDA, we will forego on discussing the implementation of these steps here.

As the above demonstrates the dynamic generating and managing of business collaboration designs using rules can be implemented using standard rule inferencing. It should be noted though that the required rule engine is more sophisticated than a normal rule engine. The typical rule agenda is replaced by an agenda of solutions, where each solution possibly consists of multiple rules. Also, the inferencing mechanism must be able to support non-monotonic reasoning on the basis of prioritization. Moreover, it must be able to deal with modalities to accommodate compensating behavior. Furthermore, the control mechanism must be able to verify changes it makes to the working memory with regard to the correctness, consistency and completeness of the memory state (based on the available control rules). We will see how a rule based business collaboration system can be constructed around such rule engine in the next section.

7.2 Icarus

In the introduction of the previous section we motivated the adoption of a service-oriented approach for the integration of rules and business collaborations. Subsequently, we explained the concept of rule engines and described different forms of inferencing. We also outlined the requirements for the rule engine needed to support the proposed design and

management algorithms. In this section we describe a prototype implementation of a system that centers around such rule engine to facilitate rule based business collaboration design and execution in a loosely coupled manner. The prototype, **Icarus**[1], was designed such that it meets the requirements identified in the introduction of this chapter. Specifically, **Icarus** focuses on the realization of the first three and last criteria, i.e. the rule based development, simulation and management of business collaborations. Due to limited time and resources the runtime generating and carrying out of business collaboration designs is currently not supported. However, in order to be able to facilitate this in the future the current prototype was created via evolutionary software prototyping. In such prototyping the idea is to build a very robust prototype in a structured manner and then constantly refine and improve it; in contrast to throwaway (or rapid) prototyping in which a software application is developed in an informal manner, evaluated and subsequently 'thrown away'. By adopting evolutionary prototyping we will be able to avoid having to start from scratch again in order to incorporate support for the aforementioned runtime generating and carrying out of business collaboration designs.

Having said that, **Icarus** is a JAVA based prototype for the development and management of rule based business collaborations. **Icarus** builds on several open source JAVA libraries, specifically on the XOM library for XML processing, the logging library by Apache for logging, and the OO jDREW reasoning library for rule reasoning.

Icarus is grounded on the conceptual architecture depicted in Fig. 7.3.

As can be seen in the figure the architecture of **Icarus** consists of six main components: **User Interface (UI)**, **Design Schema Manager (DSM)**, **Design Schema Analyzer (DSA)**, **Design Generator (DG)** and its supporting engine **OO jDREW Rule Engine**, and **File Based Repository (FBR)**. Using these different components **Icarus** enables developers to design and manage business collaborations as follows:

1. **Specify design schemas**

   To create (and edit) design schemas for their business collaborations developers of organizations like `Garage Inc` interact with **DSM** via **UI**. To this end **UI** provides a dialog based project environment (illustrated in Fig. B.1 of Appendix B showing the design schema project for **Garage Inc**'s business collaboration with **Lee C.S**). When specification is done outside the context of a specific business collaboration, then the development by e.g. `Garage Inc` will be limited to defining model schemas for its private processes and corresponding protocols. When at some point `Garage Inc` wishes to come to an agreement with `Lee C.S`, then developers from both organizations can come together to define such specifics using **DSM** via **UI**. Developed design schemas are stored by **DSM** in **FBR**, which provides a file based storage facility. **DSM** also facilitates design schema loading from this repository to allow editing of existing schemas.

[1] So named after the son of Daedalus who is famous for his death by falling into the sea when the wax holding his artificial wings together melted.

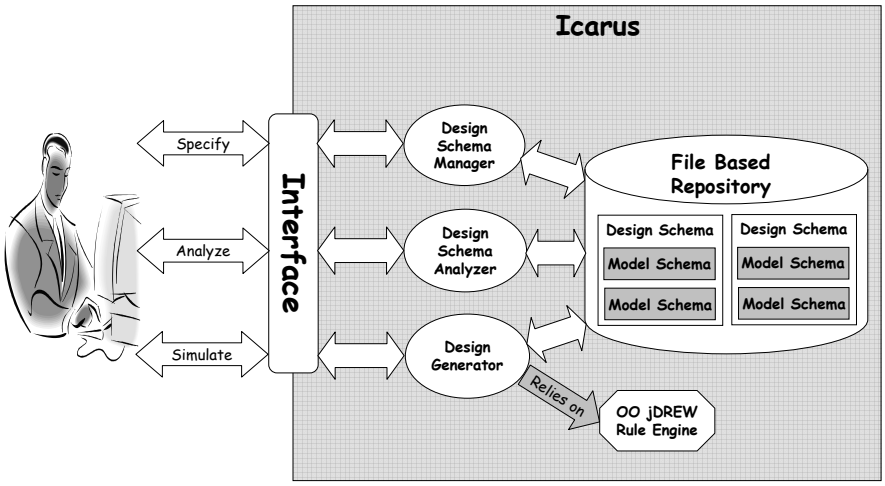

Figure 7.3: Conceptual Architecture For Icarus

2. **Analyze design schemas**

   During the design process developers can make use of the functionality provided by **DSA** to analyze developed design schemas to ensure that they are complete, consistent and correct. For example, made agreements can be analyzed to verify compatibility of the rules in the schemas underlying these agreements with those applicable to the protocols of `Garage Inc` and `Lee C.S`. Concretely, **DSA** supports checking for ambivalence, circularity, deficiency and redundancy in a design schema (conform the discussion in section 6.1 of Chapter 6). **DSA** presents any detected anomalies to the developer, who can subsequently resolve them. A screen shot involving **DSA** is provided in Fig. B.2 of Appendix B. The picture shows the result of the examination of **Garage Inc**'s design schema describing the conditions applicable to its collaboration with **Lee C.S**. As can be seen its policy for `car repair information` is currently containing several rules with redundancy as well as missing a number of rules.

3. **Generate designs**

   During the development of its business collaboration schemas `Garage Inc` can use **DG** via **UI** to simulate the design of business collaborations in accordance with the specified BICM elements and policies. **DG** constitutes a sophisticated rule engine that implements the Business Collaboration Design Algorithm (discussed in section 6.2.2 of Chapter 6). For the actual forward and backward reasoning with the

available rules **DG** relies on the **OO jDREW Rule Engine**. The OO jDREW rule engine (Ball, 2006) is an open-source, object-oriented implementation of the RuleML rule specification language (RuleML Initiative, 2006). RuleML, short for Rule Markup Language, is an XML based standard for rule specification currently under development by participants both from industry and academia. The advantage of using RuleML is that it provides a standard way of defining rules. It also unifies a plethora of industry standards allowing developed policies and rules to be deployed, executed and exchanged between different rule systems (like JESS (Sandia National Laboratories, 2006), Mandarax (Mandarax Project, 2006) and ILOG JRules (ILOG, 2006)).

Concretely, after the developer has set the initial business collaboration design and the to-be-used design schema(s) **UI** provides these to **DG**. **DG** subsequently applies the available policies to expand the design in accordance with the requirements imposed by the relevant completeness rules. When any problems are observed (like missing rules or unresolvable rule conflicts), **DG** communicates with developers via **UI**. Developers in response can return to the specification of the design schema and make the necessary changes (which are then stored in **BFR** by **DSM**). This is useful for organizations to assess the manner in which they specified their private processes and protocols as well as the mappings between them. For example, `Garage Inc` can test under different conditions whether its private processes support its protocols. This allows `Garage Inc` e.g. to find out if its car repair process is consistent with its protocol exposed to `Lee C.S` as well as to `Europ Assist`. When no more information can be derived and thus user interaction is required, **DG** returns the expanded design which is then displayed by **UI**. An example of this is shown in Fig. B.3 of Appendix B, which portrays part of the design generated for **Garage Inc**'s internal business process at strategic level.

4. **Manage existing designs**

   As business collaborations are carried out, it is possible that their underlying design schemas change. For example, `Garage Inc` wants to change its rules concerning when to get approval from `Lee C.S` concerning a car repair estimate. To this end the responsible developers from `Garage Inc` (i.e. the assigned rule steward(s)) interact with **DSM** to make the required modifications, where the consequences of these modifications can be analyzed with the help of **DSA**. Recalling from section 6.3.2 in chapter 6, the consequences of these changes are such that negotiations with `Lee C.S` will be necessary to maintain the consistency of the existing agreements. Here as well the responsible rule stewards will use **DSM** and **DSA** to make adjustments. To assess what the impact of changes would be on existing business collaboration designs, Icarus will enable developers to first generate a design conform the old design schema using **DG**, and then replace that schema with the updated schema. **DG** in response will adjust the design and highlights any made changes. To this end **DG**

implements the Business Collaboration Management Algorithm (described in section 6.3 of chapter 6). Once all changes have been finalized the developers of `Garage Inc` can store the modified design schema in **BFR**. Fig. B.4 in Appendix B displays an example of the effects of a design schema change on an existing business collaboration design; where the design generated in the previous step is changed based on a design schema of another business collaboration. The result, as the figure shows, is that the updated design is completely empty.

This concludes the discussion of the developed **Icarus** prototype. For documentation about the prototype as well as available downloads the interested reader is referred to (Orriëns, 2007). In the next section we will describe the laboratory experiment we conducted with **Icarus**.

7.3 Laboratory Experiment

To test the developed prototype we subjected it to a laboratory experiment comprising examples from the `AGFIL` case study. In the experiment we considered only part of the case study, specifically the part also used for illustrative purposes in the dissertation thus far. To recall, in the case study we focused on the interactions between `Garage Inc` and `Lee C.S` for the exchange of car repair information (as displayed in Fig. 4.3 in Chapter 4). These interactions involve several concepts such as `car repair information` and `Garage Inc` on strategic level, `receive customer file` and `estimate reported` on operational level, and `car repair service` and `report estimate request` on service level. They also involve many rules such as "if estimate is higher than $500, then get approval for car repair" and "if car repair report is to be exchanged, then a repair estimate request is to be sent". For convenience sake, we assume here that each modeling element has a policy containing a single policy alternative; that is, there is only one set of rules governing the specification and combination of the element.

Having defined the scope of the laboratory experiment we then proceeded to express several of these different modeling elements and their rules using **DSM** in Icarus. Concretely, this entailed the creation of a partial design schema in Icarus' dialog based project environment. The results are shown in Fig. B.1, which displays the created project. The developed design schema was also stored in the BCRL executable language format. This resulted in a project folder containing the different model schemas describing the different models in the business collaboration design (which can be found at (Orriëns, 2007)). One observation in this process was that the definition of the different modeling elements (and particularly their policies), requires substantial time and effort (this is also the reason why we only created a small part of the design schema). In part we feel this is inherent to the fact that the considered interactions are complex and thus are not rapidly captured. In part though there is no doubt that replacing the dialog based interface for creating design schemas by a visual drag and drop environment will be highly beneficial to the design experience. Also, we expect that re-using existing design schemas (or parts thereof) will

quicken the design process; something already illustrated by the fact that in the laboratory experiment we re-used a large set of pre-defined domain independent control rules (to check for generic completeness, correctness and consistency errors).

After having developed the initial design schema, we then verified the result using **DSA**. Concretely, we checked for ambivalence, circularity, deficiency and redundancy in the design schema. To illustrate their workings we intentionally made the policy for `supply repair information` in `Garage Inc`'s strategic agreement incomplete and redundant. As a result **DSA** found several errors, as shown in Fig. B.2 of Appendix B. This example analysis showed the usefulness of **DSA** in the development and management of design schemas in relation to the relatively small AGFIL schema, helping us to for example identify missing rules. We feel that for the definition of more extensive design schemas the benefits of **DSA** will be even greater; as in such schemas it becomes increasingly difficult to avoid accidentally defining anomalies (particularly ambivalence and circularity anomalies, whose testing we also experimented with). A way to improve **DSA** would be to replace the current text based result representation with a graphical one, which allows developers to immediately open and edit "problematic" rules. Another possible improvement is the usage of visual representations of found anomalies, especially for ambivalence and circularity issues. Finally, currently **DSA** provides limited explanatory power concerning the reason of anomalies (not greater than the reasoning displayed on screen). For providing better insight into detected problems to developers a more extensive reasoning display could prove useful.

We subsequently remedied these errors and continued onwards to then generate a design based on the given design schema using **DG**. To this end we first set the input design for the simulation. In the example scenario we take as a starting point the receipt of a `damaged car` resource, after which we determine the appropriate response. In terms of the functionalities offered by **DG** this means that we define the modeling elements needed to define the characteristics of this resource. After we defined these modeling elements, we ran the simulation process. Based on the described input and defined rules **DG** produced a partial design. Fig. B.3 of Appendix B shows a snippet of the result of the simulation process. We also stored the resulting design in its XML based variant conform the BCIM XML schema. One area of improvement arising from our experiences with **DG** is the display of the resulting design. Although it is graphical in nature, a more advanced representation would be more insightful. Additionally, **DG** currently allows to view what rules underly the different modeling elements in a generated design. However, as of yet it does not offer the possibility to gain detailed insight in its internal reasoning process (beyond that offered by the log viewer). Such view of **DG**'s reasoning process would greatly enhance the developer's understanding of a simulated design. Lastly, although the current algorithm works properly in the sense that models are generated, it does not generate designs in a fast manner. It would be interesting to optimize the algorithm's efficiency (in particular when considering generating designs on the fly during runtime).

As a last phase of the experiment we tested the **Icarus** prototype's functionality for managing business collaboration designs. We did this by making changes to its underlying design schema. Concretely, we used **DSM** to change the developed design schema to one

orthogonal to the existing one. We also performed a second test by modifying the rule of `garage repairer` concerning when to ask for approval to `consultant`. In both cases we generated a design based on the old design schema. After that we opened the management interface, where we indicated that we wished to use the modified design schema as the new underlying design schema. Next, we let **DG** simulate the effects of the changes in this schema on its current design. The result of the first test was that **DG** generated a new, empty design based on the new schema. In the second test the generated design looked similar to the existing one; however, it differed in the sense that the design now indicated that rather than asking for approval `Garage Inc` now will automatically proceed to conduct repair of the car. Fig. B.4 of Appendix B shows an illustration of the design management interface of **Icarus**. For further improvement and/or optimization of both the BCMA and its implementation more tests are necessary. Also, an experience gained from the tests was that a more detailed insight in the reasoning of **DG** will (other than that provided by the log file) will be beneficial to developers in assessing how their changes exactly affect the existing design.

7.4 Discussion

In this chapter we argued for the usability and implementability of a rule based approach for business collaboration development and management. We started by showing how the algorithms for generating and managing business collaboration designs (as defined and discussed in Chapter 6) can be implemented using standard rule engine technology. The usage of a rule engine allows us to adopt a service-oriented integration of designs and their rules, as such maximizing the degree of manageability of these rules. The idea of using service orientation to integrate rules with business collaborations is promoted for example by (Rosenberg and Dustdar, 2005). It is also visible in many products from industry. For example, the ILOG JRules (ILOG, 2006) can be integrated with other applications via decision services. The Haley Rule System (Haley Inc, 2006) can also offer access to its rules via services. QuickRules (Yasu Technologies, 2006), a rule system from YasuTech, allows its rules to be deployed as a service as well. The products are furthermore able to generate code to implement rules. Surprisingly, comparable works in academia do not mention service orientation. Systems like the DYFlow prototype and the Sword tool are not based on service orientation. Rather, the rules and service compositions appear to be tightly coupled in a model driven fashion. Another distinction between industry and academia in this regard is that most commercial products offer support for both RETE based and flow based application of rules, whereas prototypes from academia tend to only utilize forward and backward chaining. As such, the proposed rule engine bears more resemblance to industry works, since we employ both flow based and RETE based rule application when generating and managing business collaboration designs.

The practicability of a service oriented integration of designs and rules is then showcased by the Icarus prototype, which enables organizations to develop and manage their business collaboration schemas and corresponding designs. Moreover, we provide several examples

of usage of Icarus, which demonstrate how the tool assists developers when defining and modifying the rules driving the design of business collaborations. It should be noted though that the prototype is currently limited in several ways: firstly, and most importantly, there is no support at this moment for the actually carrying out business collaborations. Such support requires the Icarus architecture to be extended to include some form of business collaboration engine. Secondly, the user interface provided by Icarus is dialog based. For ease of development we feel that a graphical composing and editing of design schemas (utilizing the visual representations introduced in Chapter 4) will be easier to learn and use for developers. Thirdly, the different components in Icarus are relatively tightly coupled. From the point of reusability it seems preferable to integrate the components in a service oriented manner, where interactions between them are facilitated by middleware (for example an enterprize service bus). This will make it possible to easily add and remove components as such increasing the overall flexibility of the rule based business collaboration system.

In summary, in relation to the seventh and final research question identified in section 1.6 of Chapter 1 this chapter provides as follows:

1. We started with an exploration on implementation of the rule based business collaboration approach. We explained how through the usage of standard rule engine technology we can enable a service-oriented integration of rules into business collaborations, which allows us to maximize the degree of manageability of rules. We then demonstrated how an (enhanced) rule engine can support the generating and managing of business collaborations based on the rule inferencing mechanisms of forward chaining, backward chaining and flow based rule processing.

2. Then, we described the developed prototype Icarus, which provides an implementation of the proposed rule based approach for business collaboration. We explained how organizations can use Icarus to develop and analyze design schemas for their business collaborations as well as generate and manage designs based on those schemas. We also discussed several examples of creating and changing business collaboration design schemas and designs. Although Icarus has several limitations as mentioned above and is thus not yet fully operational, we feel that the prototype does provide sufficient functionality already to showcase the implementability of a rule based business collaboration development and management approach.

With the above answers we have addressed all of the research questions identified and defined in Chapter 1. Therefore, in the next chapter, Chapter 8, we will overview the presented research as a whole and illuminate its merits as well as its weaknesses in relation to the set out research goal and objectives.

Chapter 8

Conclusions

Reasoning draws a conclusion, but does not make the conclusion certain, unless the mind discovers it by the path of experience; Roger Bacon

A conclusion is the place where you get tired of thinking; Arthur Bloch

The business climate in which Information Technology (IT)-minded organizations have to operate nowadays is characterized by rapidly changing market conditions, new competitive pressures, new regulatory fiats that demand compliance, and new competitive threats. All of these factors and more put pressure on organizations to be able to quickly adapt their IT infrastructure in support of new business models and requirements. Otherwise, organizations will find themselves ill-equipped to function in a world that at its core relies on semi-automated, complex electronic transactions. Moreover, due to the trend of outsourcing everything but core competencies organizations are driven more and more towards cooperation with other organizations i.e. business collaborations. The realization and maintenance of such business collaborations across independent organizational boundaries and their systems is challenging as it requires linking the elements of individual businesses together into a cohesive whole. In the presence of the dynamicity resulting from the changes indicated above the challenge then becomes to maintain consistency for each partner in the collaboration as well as consistency for the collaboration as a whole. Specifically, we identified flexibility and formal adaptability, and dynamism, and undefined adaptability as the different types of dynamicity that must be supported in order to realize dynamic business collaboration development and management; where these represent expected and unexpected changes during design and runtime respectively.

In this dissertation we presented a rule based service oriented computing (SOC) approach to realize such dynamic yet consistent development and management of business collaborations, which we introduced, motivated and outlined in Chapter 1. In a service oriented computing approach the development and management of business collaborations constitutes the standardized integration of heterogeneous systems and applications belonging to different organizations through service composition. This potentially makes SOC based solutions very flexible as organizations can realize their business processes and col-

laborations in the most optimal manner by selecting and combining the most suitable and economical services, where these services may be self maintained or offered by other organizations. Unfortunately, current SOC based solutions are too narrowly focused manner concentrating on the technical aspects of business collaborations. As such, they lack the ability to capture the relation between the policies, business rules, information and processes of an organization and the manner in which organizations use their services. Additionally, service composition based on existing technologies and standards is very much a manual process. As such, modifications to service composition based business collaborations are difficult to manage both in terms of their compliance with the modified requirements as well as the consistency of the resulting business collaborations.

In contrast, characteristic of the presented rule based approach is that it builds on the premise that in the development of business collaboration, technology is secondary to the policies, business rules, information, and processes of organizations that use and create the information and the services. These policies and rules are basically statements that tell organizations what to do or not to do while conducting their business. As such, they are fundamental for the governing of the behavior of organizations. The main idea behind the rule based approach is then to separate the rules that govern business collaboration from their actual designs, and then use these rules to drive the development and management process. This makes the rules accessible and more easily manageable, in turn positioning them for change and reuse. It also gives organizations more control over their behavior particularly in the light of changing requirements. Postponement of design until runtime allows a business collaboration design to shape itself to the specific circumstances of the collaboration as it is carried out thus ensuring maximum dynamicity.

Concretely, we advocated a modularization approach concerning the context in which business collaboration takes place, where the purpose was to reduce the inherent complexity of collaboration development and management. We also argued for a model based approach to accurately and completely describe the business collaboration context, aiming to enable organizations to capture their cooperations with others in the form of models that constitute the basis for runtime carrying out of business collaborations. Furthermore, we promoted a rule based approach in which rules are used to drive and constrain business collaboration development and management, with the underlying idea to mimic the role of rules in organizations by using them to guide and govern the behavior of business collaborations at runtime. Derived from this proposal we identified seven research questions as well as four main research criteria. In the following we will first discuss the answers we obtained for these research questions in section 8.1. After that in section 8.2 we will evaluate the solution grounded upon the found answers against the criteria defined in section 1.7 of Chapter 1.

8.1 Research Results

To recall from section 1.6 we defined the following research questions: 1) what is the current state of the art in business collaboration development and management; 2) what is

business collaboration and in what context do collaborations take place; 3) how can business collaborations be best represented; 4) how can we capture business collaborations in the form of rules; 5) how can we then administer and apply these rules;6) how can we manage changes to the rules and business collaborations; and 7) how can we implement a rule based business collaboration approach? Now, an answer concerning the state of the art on dynamic business collaboration was obtained as a result of a literature review as presented in Chapter 2. The main conclusion that was distilled from this review was that there are no solutions that present a cohesive and exhaustive approach for dealing with change in business collaboration. Rather, the available solutions all tackle only part(s) of the problem. This became first apparent in the analysis of the research that has been done thus far on the context in which business collaborations take place. This research has led to the notion of separation of concern to reduce complexity by focusing on parts of collaborations while retaining the ability to put these parts together in a cohesive whole. However, although three separation of concerns are identified in literature (being aspect, level and part) none combine these into a single and cohesive contextual framework. The development of the Business Collaboration Context Framework (BCCF) described in Chapter 3 fills this gap by providing such framework, giving organizations the means to reason about the entirety of their business collaborations as well as about specific parts of these collaborations. Illustration of the usability of the BCCF was done in context of the AGFIL case study, which describes a complex multi-party scenario. Criticism of the BCCF may lie in the fact that it limits itself to adoption of only three layers of abstraction. Compared to some related works this can seem insufficient, most notably the Zachman Framework which defines six. However, as we argued in section 3.4 of Chapter 3 already this choice has been consciously made, because inclusion of six levels in the BCCF would have spiralled the complexity of the rule based model based approach out of control.

The representation of business collaborations was addressed in Chapter 4. Following the dominant trend in the literature we opted for representation of business collaborations through the usage of models. Such modeling in relation to the developed BCCF is about capturing the parts at different levels and different aspects for each individual business collaboration to capture business processes, business protocols and/or business agreements on different layers of abstraction. As a result, several models can be generated based on the level and the aspect they represent. Herein lies the reason of our hesitance to introduce more levels in the BCCF, as this would have necessitated the need to develop more models and corresponding mappings making the modeling effort involved in designing business collaborations overly complex and extensive. The models and mappings that we defined and illustrated using the AGFIL case study build where possible on existing solutions found during the literature review in Chapter 2. However, the model based approach extends these solutions in the sense that we put them into a cohesive whole as well as make their interdependencies explicit using mappings. Moreover, we defined a generic Business Collaboration Information Model (BCIM) with which the entire context of business collaborations can be described in terms of a single, formal meta-language rather than a scala of separate and isolated languages. The BCIM to this end comprises of five types of so-called modeling description atoms, being context, element, property, link and attribution.

Individual models have a context and comprise of elements, properties and links; whereas individual mappings comprise of attributions between elements of different models. The main weakness of the model based approach in our view currently lies in its limited support for the specification of advanced requirements for example regarding quality, payment and security. We explained how specification of such requirements can be facilitated, however, additional work in this area will be required to more fully capture the diversity of business collaboration requirements. For illustrative purposes we presented and discussed models and mappings between these models for the AGFIL business collaboration.

After that we submitted a rule based approach for dynamic business collaboration to meet the requirements regarding dynamicity and consistency in Chapter 5 and 6. In this approach the necessary dynamicity is established through the usage of so-called derivation rules that express the peculiarities, originality and values of individual organizations. These rules are used to drive their development and management, i.e. the creation and modification of the different models and mappings introduced and defined in the model based approach in Chapter 4. At the same time the consistency of designs is maintained through the usage of so-called control rules, which express the constraints that organizations impose on their business collaborations. Concretely, derivation and control rules together form the policies of BCIM elements, where such policies constitute the rules applicable to each individual BCIM element. BCIM elements and their policies are themselves combined into model schemas, which describe the building blocks for the design of the different processes, protocols and agreements and the rules under which these building blocks are to be combined. A design schema was then defined as a collection of model schemas, where each model schema covers a specific level/aspect combination in the BCCF. A design of an individual collaboration following the rules in this design schema was next specified as an interpretation of the schema. Depending on its complexity each design schema may have multiple possible interpretations varying with the circumstances of the individual collaboration.

In this rule based approach the development of business collaborations by organizations becomes a matter of defining design schemas. First organizations will independently develop model schemas underlying their private processes and published protocols. Next, once they wish to collaborate the organizations will define the rules governing their cooperation, i.e. constituting the agreements between them. Such shared model schemas enable organizations to assess the affects of changes to the agreements on their protocols and processes (and vice versa changes to their processes and protocols on their made agreements). This in turn empowers them to trace the impact of changes in an agreement to their private process and possibly to their agreements with other parties. As such, organizations are capable of developing and managing their collaborations with multiple parties in a consistent and dynamic manner. Subsequently, at runtime organizations generate designs for individual business collaborations in an incremental manner on the basis of their developed design schemas. By defining appropriate derivation rules organizations can steer an individual business collaboration based on the specific circumstances of its environment. This provides support for changes in the flexibility and formal adaptability categories, where different rules will be applied to handle different circumstances. Organizations manage

their business collaborations by making changes to their underlying design schemas like the introduction of new derivation rules or the modification of existing derivation rules. These changes are then effectuated into the designs of running business collaborations, where possibly existing work is undone and compensating activities are carried out. In this manner the usage of rules facilitates dynamism and undefined adaptability. All the while during design and runtime the consistency of the newly defined and modified designs is ensured through the usage of the specified control rules.

To develop the rule based approach we first analyzed the characteristics of rules in business collaboration in Chapter 5, taking matters such as non-monoticity, modality and rule grouping into account. In relation to existing works this makes that the rules that are dealt with much more expressive and complex. We also identified a wide variety of different business collaboration rules in a two-dimensional classification along function and location, which provided us with an understanding of what rules play which role in business collaboration. The functional classification gave insight into how business collaboration rules can be expressed in terms of standard rule terminology (being terms, facts and rules) as well as into the different purpose that derivation rules and control rules serve. The location classification made clear what types of business collaboration rules are necessary to generate all the models and mappings in a business collaboration design. The three-tier Business Collaboration Rule Language (BCRL) that we subsequently developed allows the specification of all these different rules in terms of a generic conceptual model that builds on the BCIM. By using different representations the BCRL can describe specified rules in a way suitable for business, formal or executable purposes. The BCRL by and large follows existing literature where possible while introducing new concepts if necessary. Most noteworthy in the latter regard are the representation of non-monoticity, modalities and versioning on the one hand, and the definition of a rich policy language on the other. This makes the BCRL a more complete and rich language in comparison to other current rule languages. To exemplify the introduced BCRL we employed illustrations from the AGFIL case study.

Armed with this knowledge we then explained in Chapter 6 how, based on specification of the different types of identified business collaboration rule, the dynamic development and management of business collaborations is facilitated. We first analyzed and proposed possible ways for organizations to create and manage their design schemas, that is, the policies and rules applicable to their business collaborations. A majority of works in existing literature assumes that these policies and rules are and remain accurate by themselves. However, when we consider the magnitude, diversity and complexity of the rules that organizations follow, the falsity of this assumption quickly becomes apparent. Without proper mechanisms to help them developers will be at a loss when trying to ensure that developed rules and policies are and remain consistent. We tackled this problem by looking at well known issues in rule systems and subsequently develop mechanisms that provide organizations with the means to detect and resolve problems with their defined rules. Specifically, we developed mechanisms for the detection of redundancy, inconsistency, circularity and deficiency anomalies. These mechanisms were partially inspired by existing solutions, however, we extended them significantly to be able to cope not only with the

possible non-monoticity and modalities of rules, but also with different versions of rules as well as with rules that are in different states of their life cycle. We illustrated the benefits of the mechanisms using examples in context of the AGFIL case study.

We then developed and formally defined the relationship between business collaboration rules and business collaboration designs in terms of model-theoretic truth using model-theory and model-theoretic semantics. Shortly states, such truth entails that a rule is true if and only if it is true in the well-founded model, and consequently the rule is false if and only if it is false in that model; where the well-founded model is the interpretation in which everything that is necessarily true and not true according to the rules is true c.q. not true in the model. After that, based on this definition of model-theoretic truth we developed the notion of conformance, which allows organizations to verify if a design is conform the requirements in its underlying design schema as well as that this design is consistent. Following, we developed and demonstrated a generic Business Collaboration Design Algorithm (BCDA) consisting of five main steps that uses conformance to facilitate development and management in a dynamic yet controlled manner through the application of rules, and with which flexibility and formal adaptability can be accommodated for in an uniform manner. In comparison to other rule based solutions this algorithm offers much more support for dynamicity due to its generic nature and the range of different rules it can apply (as we showcased in context of the AGFIL case study). The algorithm also ensures that developed designs are and remain consistent, an issue also largely ignored by other rule based approaches.

A justified critic on the BCDA is that the offered degree of dynamicity and consistency does not come without a cost. With every rule based expansion of a business collaboration design the appropriate rule(s) must be found and applied, and the result must be verified for consistency. This will be less efficient than using pre-defined designs, and in this sense a trade-off of dynamicity and consistency versus performance must be made by organizations when considering in what way to develop and manage their business collaborations. However, as business collaborations tend to be long running in nature we feel that in most situations this decreased level of efficiency will not result in delays that are unacceptable to organizations. Moreover, when organizations limit the amount of possible variations for their business collaborations in the corresponding design schemas, the generating of designs using the BCDA is much simpler and efficient. As such, organizations can also opt to adopt the rule based approach for their relatively static business collaborations, where they can then introduce more dynamicity if so desired. We concluded with an analysis of how the modification of a design schema affects already existing business collaboration designs. Based on this analysis we defined the Business Collaboration Management Algorithm (BCMA) which facilitates the (semi-)automated handling of dynamism and undefined adaptability changes to running business collaborations. Concretely, as we demonstrated using the AGFIL case study, new elements can be added to a design schema with their associated policy, existing elements and their policies can be removed, and changes to existing policies can be made (such as the addition of new rules and modification/deletion of existing rules). These changes are then incorporated into existing business collaborations conform the BCMA, which ensures the consistency of the

resulting designs. This enables organizations to adjust their requirements and effectuate these into running business collaborations whilst at the same time be assured that these collaborations remain consistent and valid.

Next in Chapter 7 we showcased the practical feasibility of the rule based business collaboration approach. We first discussed the key issue for implementing rule based business collaboration, being the integration of rules and business collaborations. We looked at several options before opting for a service oriented approach using a rule engine as this provides the greatest degree of flexibility and manageability in the usage of rules in business collaboration development and management. We then explained how such rule engine takes care of the retrieval and administering of rules to generate and manage designs by implementing the BCDA and BCMA algorithms. This has the advantage that the development and management of business collaboration rules is separated from the actual running and evolution of individual business collaborations. After that we implemented a rule based business collaboration system in the prototype Icarus. Icarus is a JAVA based tool, which facilitates the specification and analysis of business collaboration schemas as well as the generating and management of business collaboration designs (as demonstrated by several examples). As observed at the end of Chapter 7 the prototype is limited in some fashions, most notably the fact that currently the actual carrying out of business collaborations is not supported. We are confident though that facilitating such support is not an insurmountable problem, particularly as the prototype was developed using evolutionary prototyping.

8.2 Result Evaluation

In the previous section we summarized and analyzed the results we obtained in answering the research questions we raised in section 1.6 of Chapter 1. In this section we turn our attention to evaluating the resulting solution for dynamic business collaboration development and management against the criteria stipulated at the end of section 1.7 in Chapter 1. To recall, we defined four main criteria there revolving around extensiveness, manageability, verifiability and usability respectively. The business collaboration development and management solution described in this dissertation meets these criteria as follows:

- **Extensiveness**

  With regard to the criterium that the proposed solution must be extensive enough for organizations to capture the intricacies of their business processes, protocols, agreements and the interplay between them, we can conclude that based on the results described in Chapters 3 and 4 this can be done in a comprehensive and cohesive manner; placed in context by the aspects, levels and parts identified in the Business Collaboration context Framework (see sections 3.1 through 3.3). The different business processes, protocols, and agreements in a business collaboration are captured in individual models at strategic, operational and service level (see sections 4.1 through 4.3). Dependencies among levels are expressed using vertical

mappings (see section 4.4), whereas dependencies among processes, protocols, and agreements are defined using horizontal mappings (see section 4.5).

Models themselves consist of modeling elements which express the different parts. By enriching modeling element definitions with additional characteristics advanced requirements such as security and quality can be expressed. It should be noted that in this dissertation we only demonstrated this for exemplary parameters. However, due to the uniformity of the Business Collaboration Information Model (see section 4.6) underlying the different models and mappings the model based approach is easily extendable. Support for other advanced requirements can be straightforwardly provided as new properties can be defined for any of identified modeling elements. Moreover, dependencies among properties at different aspects, levels and parts can be expressed and enforced using control rules.

- **Manageability**

  The manageability of business collaboration designs is provided by the proposed rule based approach for developing the required models and mappings. By enabling developers to define the basic building blocks for these models and mappings and express their constraints in policies consisting of derivation rules, the modeling process becomes a runtime activity in which a business collaboration design shapes itself to the circumstances of the collaboration as it is progressing. Since these derivation rules are explicitly defined, they become manageable (see section 5.1). Moreover, because derivation rules are employed to govern the entirety of a business collaboration, changes can easily made to any part in an uniform manner(see section 5.3), where these changes can be both fine-grained and coarse grained.

  In addition, developers can assess the impact of changes. Firstly, they can ensure that modifications do not lead to anomalies such as cyclic behavior or conflicting rules. To this end developers are assisted by the different business collaboration development mechanisms (see section 6.1). Secondly, through simulation they can analyze what the consequences of changes are when actually designing a business collaboration (see section 6.2). Furthermore, developers can keep track of changes as all rules have associated meta-data like a description, history, and etceteras. The version, and activation and expiration date date attached to each rule allow developers to create multiple variants of business collaboration designs for varying circumstances (see section 6.2.2). Versioning also enables them to apply changes only to new designs, existing designs, or both (see sections 6.3.2 and 6.3.3).

- **Verifiability**

  Whereas the manageability of business collaboration designs to a large extent comes from the explicit definition of derivation rules, their verifiability is facilitated by control rules (see section 5.3). Like derivation rules, control rules are explicitly specified

in the policies of individual modeling elements. These control rules enforce the constraints applicable to the specification of each modeling element and the manner in which it can be combined with others. Since control rules (like derivation rules) are used to constrain the whole of a business collaboration, the validity, alignment and compatibility (see section 1.2.2) can be verified in an uniform manner by checking whether the design satisfies the control rules (see section 6.2.1). Moreover, conformance of a design to its derivation rules (i.e. the rules used to create the design) can be verified in the same way (see also section 6.2.1). As such, both conformance as well as validity, alignment and compatibility can be verified.

- **Usability**

  Finally, the usability of the proposed rule based approach for dynamic business collaboration development and management is supported in several ways. Firstly, the modeling elements that are identified as the basic building blocks for business collaboration designs, closely resemble those employed in existing standards and specifications. As such, they will be familiar to developers, which will decrease the steepness of the learning curve. Secondly, the conditions applicable to each modeling element are captured in the form of rules. Rules are intuitive constructs for developers as human beings use them in everyday life. As such, they will be relatively easy for developers to employ in order to express business collaboration requirements. This is particularly the case as rules are presented to developers in a user-friendly, text-based manner. Moreover, by associating every rule with a set of meta-data developers have much more insight into the why and how of each of the expressed business collaboration requirements.

  Additionally, due to the multi-level approach the developed solution allows different types of developer to work in a jointly manner to specify and manage designs for business collaborations. For example, the modeling elements (and their conditions) that comprise the high level strategic models in an individual collaboration can expected to be defined and maintained by business managers. If needed this can be done with the help of modeling experts, where the managers give input and feedback on the designed models. Alternatively, if an intuitive, wizard like GUI is provided in which business managers are taken step by step though the design process, they can perform these activities by themselves. The same goes for process and data managers responsible for creating operational models, as well as technical experts for service models. Furthermore, these different types of developers can use the visual representations of the models for communication purposes when identifying and expressing dependencies among strategic, operational and service models.

  When it comes to the modification and management of developed business collaborations, the solution enables developers to make changes in an easy and uniform manner. By simply re-defining existing modeling elements and their policies or by

adding new ones, developers can introduce changes to designs. The impact of these changes in new designs can then be simulated following the Business Collaboration Design Algorithm. Moreover, their effects on existing designs can be assessed and analyzed conform the Business Collaboration Management Algorithm. Finally, reuse is facilitated in several ways in the developed solution on different levels of granularity. Firstly, developers can reuse entire model schemas across multiple design schemas. This allows them to link together different business collaborations; which is necessary to capture the phenomenon described in section 3.1 of Chapter 1 regarding the propagation of change from one business collaboration to another. It also enables developers to create design schemas at greater speed. Secondly, at the level of individual elements developers can partially or completely reuse policies, alternatives and/or rules through the referencing mechanism in the BCRL. Moreover, they can build hierarchical trees of policies, alternatives and rules via the extension mechanism in the BCRL.

Given the above we can conclude that the research presented in this dissertation describes a solution for dynamic business collaboration development and management that is not only extensive, but also manageable, verifiable and usable. As such, we feel the research constitutes a significant contribution to the field of dynamic business collaboration. Concretely, by achieving the set out research objectives we contributed to existing work through the development of: 1) a cohesive and comprehensive Business Collaboration Context Framework for the context in which business collaborations take place; 2) an extensible model based approach founded on a generic Business Collaboration Information Model (BCIM) to describe this context (and thus business collaborations); 3) a definition of business collaborations rules and a classification of such rules, and a three-tiered generic Business Collaboration Rule Language (BCRL) to specify policies and rules with; 4) a set of mechanisms with which policies and rules can be specified and managed in a consistent manner; 5) a generic Business Collaboration Design Algorithm (BCDA) to administer the specified policies and rules in order to develop business collaboration designs in a dynamic and consistent manner; 6) a Business Collaboration Management Algorithm (BCMA) with which the impact of rule and policy modifications on existing designs can be assessed and automatically managed; and 7) a prototype implementation called Icarus that showcases the practical feasibility of the rule based business collaboration approach. As a result, we have shown that with the presented approach organizations gain the ability to quickly adapt the manner in which they conduct their semi-automated, electronic transactions in response to new business models and requirements; a vital ability if they wish to survive and prosper in a business climate in which change is the norm rather than the exception.

Chapter 9

Future Work

Prediction is very difficult, especially of the future; Niels Bohr

The future, according to some scientists, will be exactly like the past, only far more expensive; John Sladek

As we have seen in this dissertation building and managing business collaborations that cross independent organizational boundaries and their systems is challenging especially in light of the dynamic world in which organizations operate. Organizations have to tackle the problem that in developing business collaboration, technology must work in conjunction with the policies, business rules, information, and processes that use and create the information and the services. Organizations also have to cope with the challenge of maintaining consistency for each partner in the collaboration as well as consistency for the collaboration as a whole. The resulting scope and complexity involved makes it almost inevitably that any proposed solution for dynamic business collaboration development and management will fall short in some way. Our work is no exception to that rule as reflected somewhat already in the scope limitations we set in section 1.5 of Chapter 1. Following these limitations as well as taking other considerations into account such several directions for future research can be identified.

One such direction that we expect to be of interest is the expansion of the business collaboration context, particularly with regard to its set of abstraction layers. In the proposed contextual BCCF framework three such levels are identified. However, solutions from other works like (Zachman, 1987) suggest that perhaps more levels are needed to capture all the different relevant perspectives on business collaboration; and thus to allow all the different types of people involved in business collaboration to talk and reason about these collaborations in terms that are familiar to them. Of course, extending the BCCF in this manner will necessitate the definition of new models and mappings corresponding to these levels. However, this seems to be feasible when considering the model based approach presented in this dissertation. This remains an unverified hypothesis though until it is tested in future research. Also, a clear view should be included in the BCCF on how issues such as quality and security relate to the different aspects, levels and parts. We

briefly mentioned our view on this in section 3.4 of Chapter 3, however, this view needs to be made more concrete.

Related to this is the second area in which we envision that there is fruitful work to be done, being the specification of advanced business collaboration requirements. In this dissertation we only briefly discussed how quality and security related requirements can be captured in context of the models and mappings that we developed by enriching elements with additional properties. However, this needs to be worked out in more detail. Early reports of such work can be found in (Orriëns, 2006c) and (Orriëns, 2006d). Our research would also benefit from support for the definition of other types of requirements as well, for example concerning transactional semantics, payment and billing, and legal matters. We expect that this will not only give organizations the means to describe their business collaborations in full detail, but it will also empower them to develop and manage much more expressive business policies and rules still; and thus control and manage their business collaborations to an even greater extent. Of yet though this is only a theoretical possibility which requires further exploration. Furthermore, we expect that the integration of semantic web based technologies will be beneficial. Currently we make the assumption that organizations share the same semantics, however, this need not be the case of course. Ontologies can be helpful here to enrich the model based approach such that organizations can use their own ontology to define the terms used to describe their business processes and protocols, where these are then mapped to a shared ontology that is associated with the business agreements.

The presented rule based approach may also be subject to extension, revision and refinement. Although the rules in this approach are relatively expressive compared to other works, the support for modalities does not cover all possibilities in this regard. It would be interesting to develop a generic mechanism for modality specification, combination and interpretation, and incorporated it into our approach. The BCDA and BCMA algorithms for the design and management of business collaborations respectively have proven to support the different forms of identified dynamicity while ensuring the consistency of the resulting business collaborations. However, they may require re-definition to improve efficiency, also in relation to the manner in which the algorithms are currently realized in the suggested conceptual architecture for a Rule Based Business Collaboration System. But before we can accurately make such assessment, the rule based approach should be applied in several other case studies and real life settings.

Another interesting direction for future research is that of automated support for the negotiation of agreements. In the presented rule based approach organizations have the means to test the consistency of their business protocols in relation to their business agreements with other parties using the organization's control rules. Rather than employing these control rules to detect inconsistencies in a passive manner, it seems feasible to use them in a more active manner to automatically generate an agreement conform the derivation rules underlying their protocols. The resulting agreements can then be checked for feasibility in terms of their consistency. Subsequently problem areas can be identified, where the monoticity and prioritization of inconsistent rules reflect whether they express conditions that are negotiable or not. This will bootstrap the negotiation process between

organizations as skeleton agreements can be automatically generated. Moreover, when multiple generated agreements are viable, optimization criteria can be employed to find the most suitable agreement (for example with the help of techniques from the field of constraint satisfaction).

In addition to the above, this dissertation has focused on how the development and management of business collaborations and their designs can be performed in a dynamic manner in accordance with shifting requirements while maintaining consistency of the resulting designs. We did not pay explicit attention to the monitoring of running business collaborations as this is outside the scope of the presented research. Such monitoring is necessary though since organizations must be able to assess whether their requirements are met or not as they are collaborating (e.g. to detect that a quality requirement has been violated). However, whereas the monitoring of such runtime events is outside the scope of our work, the specification of appropriate measures is accommodated for. As part of the design schema for their business collaboration organizations can define what monitoring events can occur at runtime as well as specify rules that depict how to handle these events. Then, when such events are observed to have occurred during the business collaboration, organizations can dynamically react to the violated requirements by applying the suitable rules. This will make the business collaborations of organizations much more robust and versatile.

Finally, from a research validation point of view more work is needed to test and evaluate both the usability and implementatibility of the proposed rule based approach for business collaboration development and management. In this dissertation we applied the approach in the context of a complex but theoretical multi-party business scenario. The application of the approach in real life situations will give valuable insight into its workings in a practical setting. To achieve such insight multiple case studies should be conducted and analyzed with regard to the found results. Consequently, if needed the rule based approach can be adjusted and fine-tuned to overcome any found discrepancies and/or problems. In relation to the implementatibility of the suggested approach work is required for the Icarus prototype to support the linking of the rule driven generating business collaboration designs with the actual carrying out of these designs at runtime. This will transform Icarus from a design time development and management tool suited for design and analysis to a full fleshed tool also capable of controlling and managing running business collaborations.

Appendix A

AGFIL Case Study

A problem is a chance for you to do your best. Duke Ellington

For every complex problem there is an answer that is clear, simple, and wrong. H. L. Mencken

To exemplify the ideas presented throughout this dissertation an example inspired by the case study in (Grefen et al., 2000) is used. The example describes a complex multi-party scenario, which outlines the manner in which a car damage claim is handled by an insurance company (AGFIL). AGFIL cooperates with several contract parties to provide a service level that enables efficient claim settlement. The parties involved are Europ Assist, Lee Consulting Services, Garages and Assessors. Europ Assist offers a 24-hour emergency call answering service to policyholders. Lee C.S coordinates and manages the operation of the emergency service on a day-to-day level on behalf of AGFIL. Garages are responsible for car repair. Assessors conduct the physical inspections of damaged vehicles and agree repair upon figures with the garages. These parties work together as follows: policyholders (i.e. customers) phone Europ Assist using a free-phone number to notify a new claim. The claim is received by a call handler within Europ Assist's telephone assistance department. After verification of the customer' credentials to ensure that the provided policy details are valid and the occurred loss is covered, the call handler finds an approved repairer nearest to the customer's location. The customer is notified that this repairer will arrive at the scene shortly, if necessary with a replacement car and towing service. The call handler subsequently contacts the selected repairer to notify him of the incident. If the repairer is not available, another one will be selected and contacted. The customer is kept posted of such changes by phone. Once the repairer is on its way, the call handler contacts AGFIL to inform them of the made claim.

Upon receipt of the claim a claim handler will be assigned within AGFIL. The claim handler will gather all related claim information like customer records, claim history, etc, and send it to Lee C.S After that the claim handler will fill out the claim details on a claim form, which is subsequently stored pending further developments. Lee C.S in the meanwhile has one of its consultants working on the claim. The first thing this consultant

does, is contact the garage to inquire about the status of the car. The garage has picked up the car while the previous was going on and has worked out an estimate of the car repair cost. If this cost was below $500 then the garage will have started repairs. But if the costs were higher, the consultant at Lee C.S contacts an assessor to go to the garage and check out the car for him -or herself. This assessor makes an independent estimate of the repair costs and negotiates a final price with the garage. The result of the assessment is next reported back to the consultant at Lee C.S The consultant reads the report and approves repair. An approval notification is sent to the garage, which consequently starts repairs on the car. Lee C.S' consultant also informs the claim handler at AGFIL of the final repair cost estimate upon which the claim handler incorporates the new information in the claim form. Once the garage has completed its repairs on the customer's car, an invoice is communicated to the consultant at Lee C.S The consultant checks the invoice to see if it matches the earlier received cost estimate. Once the invoice is approved, the consultant sends the invoice onwards to AGFIL. The claim handler receives the invoice and adds it to the claim form. Payment for the claim is also issued.

This scenario is illustrated in Fig. A.1 to Fig. A.3 showing the strategic, operational and service level representation of the interactions between the different parties. The reader is to be aware that the displayed models are intended for illustrative purposed only. As such, for reasons of clarity the models in the different figures are not complete. That is, we omitted to completely define the private, exposed and agreed upon behavior of each party. Moreover, we did not define all events and triggers in the operational and service models respectively.

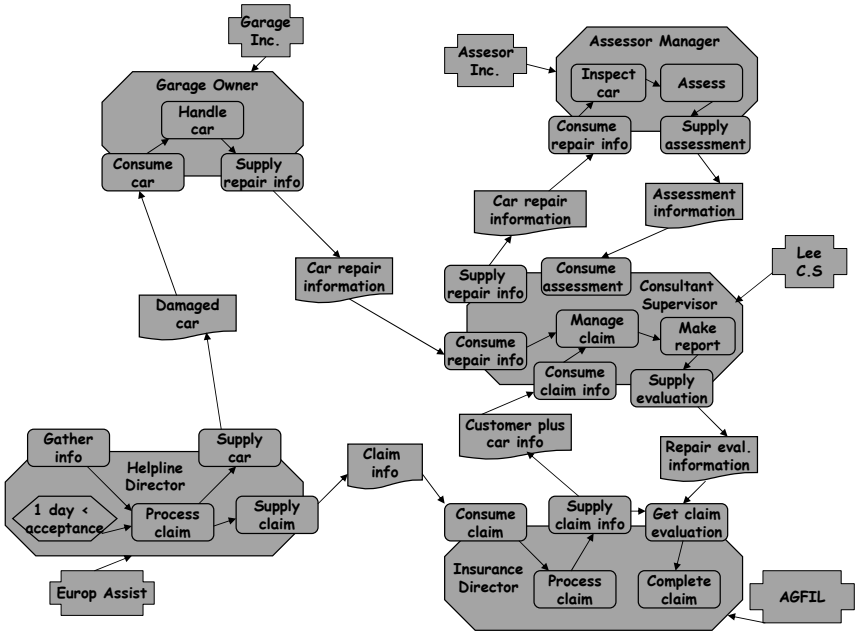

Figure A.1: AGFIL Scenario Strategic Overview

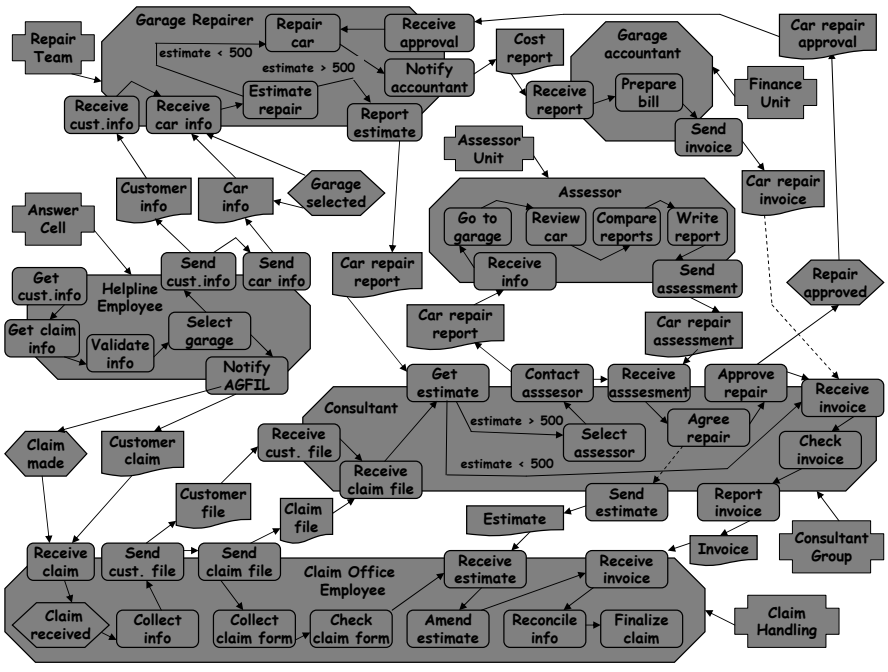

Figure A.2: AGFIL Scenario Operational Overview

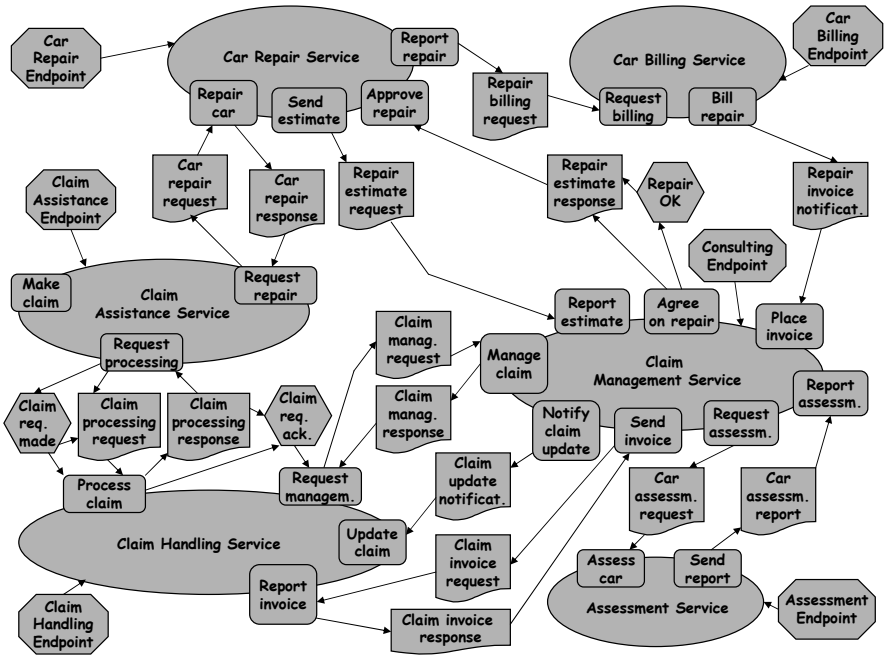

Figure A.3: AGFIL Scenario Service Overview

Appendix B

Icarus Screen Shots

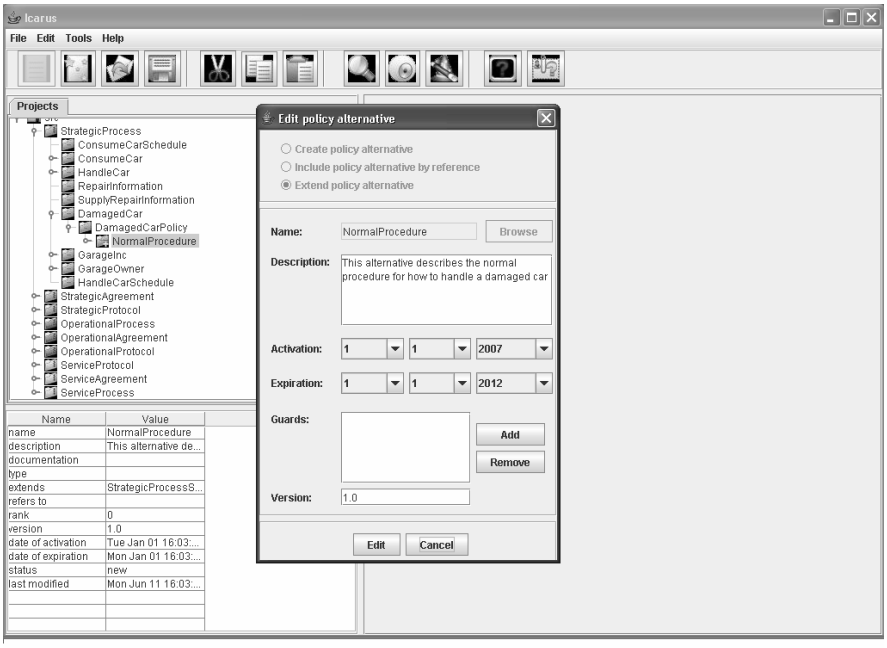

Figure B.1: Specifying A Design Schema

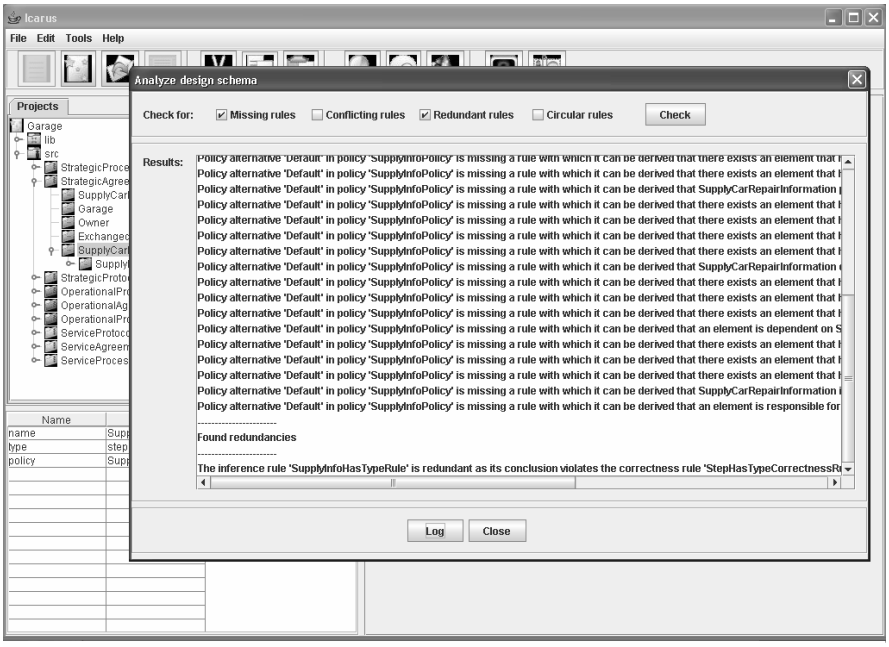

Figure B.2: Analyzing A Design Schema

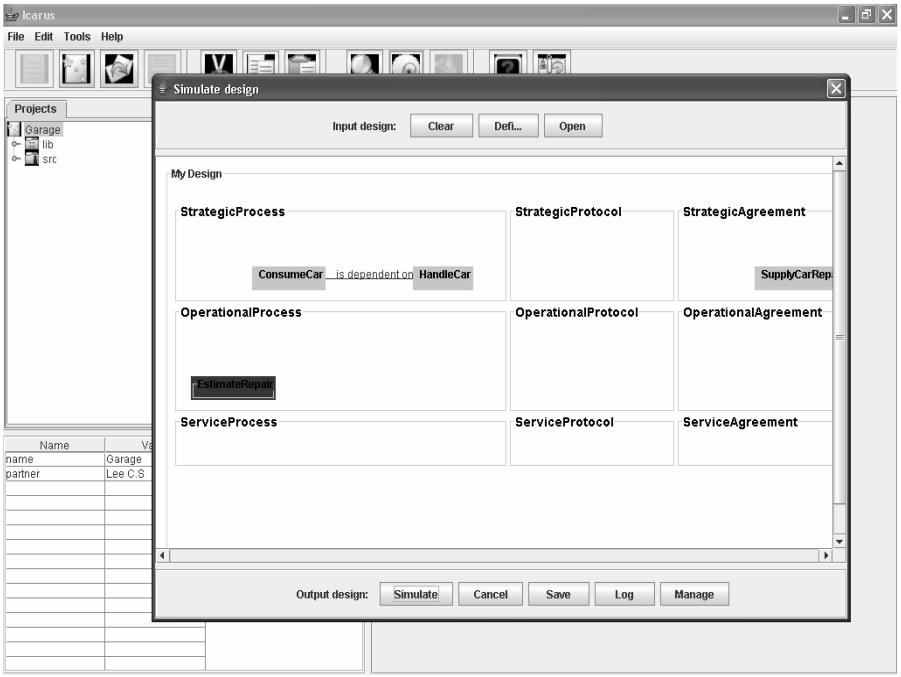

Figure B.3: Generating A Design

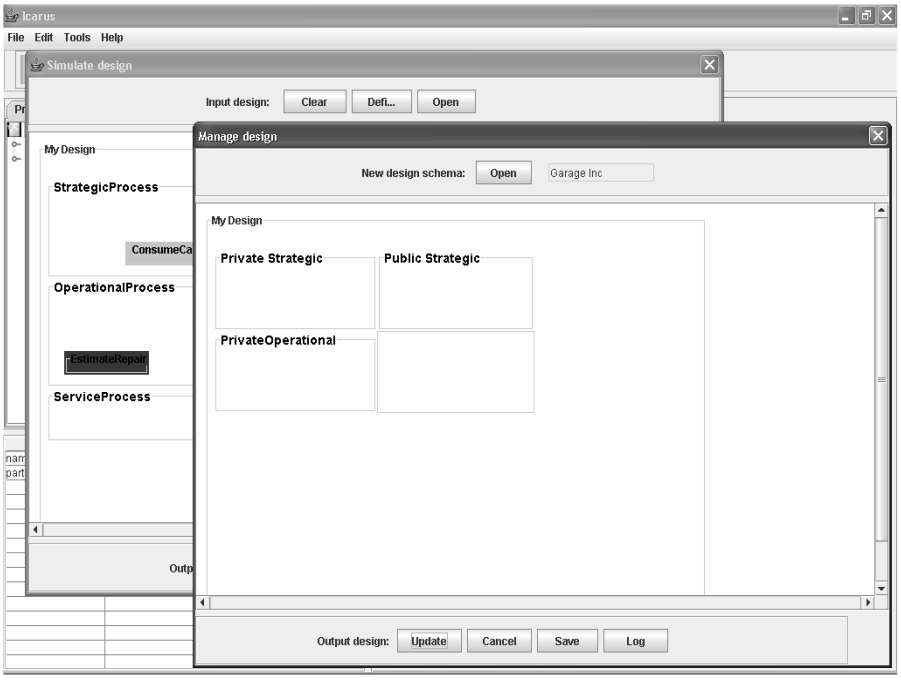

Figure B.4: Managing An Existing Design

Appendix C

Glossary

I tell you, define your terms or we will never understand each other; Voltaire

We see things not as they are, but as we are; Henry Major Tomlinson

Chapter 1

Alignment: the lack of contradictions between the representations of business processes, business protocols or business agreements at different *levels*. Two business processes, business protocols or business agreements are aligned if the *mapping* between their *models* is in *conformance* with their *control rules*.

Business Agreement: an agreement among organizations detailing the activities they intend to perform in order to carry out their *business collaboration*.

Business Collaboration: a cooperation between organizations to achieve shared goals by coordinating and linking their *business processes* exposed via their *business protocols*, and where the conditions under which they are collaborating are depicted in *business agreements*.

Business Collaboration Design: the activity of analyzing the *business collaboration* that is to be automated and the subsequent creation of a *design* of the collaboration. This *design* provides a (often simplified) view of the collaboration.

Business Process: a collection of related structural activities that produce something of value to an organization.

Business Process Automation: the process of designing and enacting *business processes* using Information Technology.

Business Process Design: the activity of analyzing the *business process* that is to be

automated and the subsequently creation of a design of the process. This design provides a (often simplified) view of the process.

Business Process Enactment: the activity of creating *instances* of a *business process design*, which are then executed to carry out the *business process*.

Business Process Instance: the instantiation of the design of a *business process*.

Business Protocol: the public behavior of an organization, also referred to as an abstract business model.

Business Rule Approach: a development methodology where *rules* are in a form that is used by, but not embedded in business process automation systems, to improve the *dynamicity* of the managed *business processes*.

Compatibility: the lack of contradictions between the *business processes* and *business protocols*, and the *business protocols* and *business agreements* of an organization. A business process and business protocol, and a business protocol and business agreement are compatible if the *mapping* between their *models* is in *conformance* with their *control rules*.

Consistency: the lack of contradictions in a *design* of a *business collaboration*. Verified via the usage of *domain independent rules* and grounded on the notion of *conformance*.

Dynamicity: the ease with which *business collaborations* can be changed during design time and runtime. Is further divided into the categories of *dynamism*, *flexibility*, *formal adaptability* and *undefined adaptability*.

Dynamism: the ability to make a *change* known at design time to a *business collaboration* by changing its *design* at runtime.

Flexibility: the ability to incorporate support for a *change* in a *design* of a *business collaboration* that is known at design time, and is known to occur at some specific point during runtime.

Formal Adaptability: the ability to incorporate support for a *change* in a *design* of a *business collaboration* that is known at design time yet is unpredictable in nature at runtime.

Inter-organizational Process: see *business collaboration*.

Intra-organizational Process: see *business process*.

Model Based Approach: an approach with which *business collaborations* can be accurately and completely described in the form of *models*.

Modularization Approach: an approach with which the inherent complexity of *business collaborations* is made manageable by slicing development and management into multiple smaller chunks.

Rule Based Approach: an approach with which the development and management of *business collaboration* is driven and constrained by *rules*.

Rule Management System: a system that supports the authoring, deployment and management of *rules*. Typically used in the context of a *business rule approach*.

Service: an independent software entity that performs functions. These functions can vary from very simple requests to complete *business processes*.

Service Oriented Computing: a paradigm based on the notion of using *service* for distributed computing and e-business processing to enable building agile networks of collaborating business applications distributed within and across organizational boundaries (also known as SOC).

Undefined Adaptability: the ability to make a *change* to a *business collaboration* by changing its *design* at runtime, which is unknown at design time and occurs unpredictably at runtime.

Validity: the lack of contradictions in a *business process*, *business protocol* or *business agreement*. A business process, business protocol or business agreement is valid if its *model* is in *conformance* with its *control rules*.

Chapter 3

Aspect: places emphasis on the different *business collaboration behaviors* that an organization exhibits in *business collaboration*. Three aspects exist in the business collaboration context being the *conversation*, *participant public behavior* and *internal business process aspect*.

Business Collaboration Behavior: the behavior of an organization in a *business collaboration*, be it private in its *business processes*, exposed in its *business protocols* or observable in its *business collaborations*.

Business Collaboration Context Framework (BCCF): a framework that provides a modularized representation of the context in which business collaborations taking place considering both *intra-organizational* and *inter-organizational processes* from business and

technical point of view.

Conversation Aspect: captures the externally visible behavior between participants in a *business collaboration*, i.e the observable behavior.

Functional Part: concentrates on how a *business collaboration behavior* is conducted.

Internal Business Process Aspect: encompasses the internal activities of participants in a business collaboration, i.e. the behavior that can not be observed by other participants.

Level: represents a layer of abstraction at which business collaboration behaviors can be observed. Three levels exist in the business collaboration context being *strategic*, *operational*, and *service level*.

Operational Level: concentrates on the activities that organizations conduct on a day-to-day basis in support of their strategic objectives.

Location Part: expresses the geographical and organizational locations at which a *business collaboration behavior* is carried out.

Part: represents a part of a *business collaboration* by depicting the *elements* in a *business collaboration behavior* that have different contexts when observed from different *levels*. Five parts exists in the *business collaboration context* being *structural*, *functional*, *participation*, *location*, and *temporal part*.

Participant Public Behavior Aspect: describes how a participant can publicly interact in a business collaboration, i.e. its potential behavior for cooperating with others.

Participation Part: concerns the participants involved in a *business collaboration behavior*.

Service Level: addresses the technical requirements of *business collaboration behaviors* describing how organizations use their IT-infrastructure within the context of their *business collaborations*.

Strategic Level: focuses on the behavior that is abstract in nature, describing the purpose and high level requirements an organization has with their business collaboration behavior.

Structural Part: deals with the materials that are used to perform a *business collaboration behavior*.

Temporal Part: pertains to time related issues of relevance for a *business collaboration behavior*.

Chapter 4

Agreement: a *model* describing the agreement made between two organizations with regard to how each is expected to behave. Captures the *conversation aspect* in the business collaboration context.

Attribution: specifies a relation between *element*s from different *models*, i.e. express *mappings* among *elements*.

Business Collaboration Information Model: the meta model underlying the different *models* used to describe *business collaborations* consisting of five types of *modeling description atom*.

Context: identifies the position of a *model* within the business collaboration context of a *business collaboration* in terms of its *level* and *aspect*.

Design: the collection of *models* and *mappings* resulting from *business collaboration design* that together describe the context of individual *business collaborations*.

Element: represents a *part* of a behavior, i.e. *structural*, *functional*, *participation*, *location*, or *temporal part*.

Horizontal Mapping: a *mapping* between two *business collaboration behaviors* describing the same *level* but at different *aspect*.

Link: expresses a connection between *elements* belonging to the same *model*.

Mapping: a representation of the dependencies among *business collaboration behaviors*, that is, between two *models*, consisting of *attributions*.

Model: a representation of a *business collaboration behavior* in a *business collaboration* consisting of *modeling description atoms* as defined in the *Business Collaboration Information Model*.

Modeling Description Atom: a basic building block with which *models* are constructed to capture the different *business collaboration behaviors*. There are five types of modeling description atom, being *elements*, *properties*, *links*, *attributions*, and *contexts*.

Operational Model: a *model* expressing part of the *operational level* of a *business collaboration*, be it an *agreement*, *process* or *protocol*.

Process: a *model* describing the private activities of an organization. Captures the *internal business process aspect* in the business collaboration context.

Property: defines a characteristic of an *element*, enriching the description of a *part*.

Protocol: a *model* describing the potential manner in which an organization can act within in a *business collaboration*. Captures the *participant public behavior aspect* in the business collaboration context.

Service Model: a *model* expressing part of the *service level* of a *business collaboration*, be it an *agreement*, *process* or *protocol*.

Strategic Model: a *model* expressing part of the *strategic level* of a *business collaboration*, be it an *agreement*, *process* or *protocol*.

Vertical Mapping: a *mapping* between two *business collaboration behaviors* describing the same *aspect* but at different *level*.

Chapter 5

Antecedent: the condition(s) of a *rule*, that is, what must be true for the rule's *consequent* to be true. Also known as the left hand side (LHS) of a *rule*, the antecedent comprises a conjunction of one or more *clauses*.

Business Collaboration Rule Language: the language used to express *business collaboration rules*, based on the *modeling description atoms* as defined in the *Business Collaboration Information Model*.

Business Collaboration Rule: an atomic, declarative, and reliable *rule* written in a language that can be understood by business people, which is intended to assert business structure or to control or influence the behavior of business collaborations by stating either what should or should not be the case. A business collaboration rule may be *non-monotonic* in which case it will be *prioritized*.

Business Language Rule: a *business collaboration rule* expressed in terms of the business language of the *Business Collaboration Rule Language*.

Business Rule: a *business collaboration rule* that constrains the *operational level* of a *business collaboration*.

Clause: a statement that constrains one or more *modeling description atoms*, optionally through negation and/or modality.

Completeness Rule: a *control rule* that makes sure that all (and not more) information that is needed to describe a *business collaboration* has been defined.

Consequent: the conclusion(s) of a *rule*, that is, what is true if the rule's *antecedent* is true. Also known as the right hand side (RHS) of a *rule*, the consequent comprises a single *clause*.

Consistency Rule: a *control rule* that ensures that the information defined to describe a *business collaboration* does not contain any contradictions.

Constraint: a *control rule* that is mandatory in nature, i.e. monotonic.

Control Rule: a *business collaboration rule* that constrains *business collaborations* in such a manner that these are and remain in accordance with the conditions imposed by the business collaboration domain. Facilitates the verification of derived knowledge.

Correctness Rule: a *business collaboration rule* that ensures that the information defined to describe a *business collaboration* has been properly specified.

Design Schema: the collection of *elements* and their *policies* that together form the building blocks for the *design* of a *business collaboration* from the point of view of an individual organization. Comprises of multiple *model schemas*.

Derivation Rule: a *business collaboration rule* expressing the peculiarities, originality and values of an individual organization with regard to how it determines in what way to conduct its business collaborations. Facilitates the derivation of knowledge.

Executable Language Rule: a *business collaboration rule* expressed in terms of the executable language of the *Business Collaboration Rule Language*.

Fact: a statement that connects *terms*, through prepositions and verb phrases, into sensible, business relevant observations.

Formal Language Rule: a *business collaboration rule* expressed in terms of the formal language of the *Business Collaboration Rule Language*.

Functional Rule: a *business collaboration rule* that steers/constrains the *functional part* of a *business collaboration behavior*.

Functional-Location Rule: a *business collaboration rule* that steers/constrains the relation between the *functional part* and *location part* of a *business collaboration behavior*.

Functional-Participation Rule: a *business collaboration rule* that steers/constrains the relation between the *functional part* and *participation part* of a *business collaboration behavior*.

Functional-Temporal Rule: a *business collaboration rule* that steers/constrains the relation between the *functional part* and *temporal part* of a *business collaboration behavior*.

Goal: a *business collaboration rule* that steers/constrains the *strategic level* of a *business collaboration*.

Guideline: a *control rule* that is not mandatory in nature, i.e. non-monotonic.

Limitation: a *business collaboration rule* that constrains the *service level* of a *business collaboration*.

Location Rule: a *business collaboration rule* that steers/constrains the *location part* of a *business collaboration behavior*.

Location-Temporal Rule: a *business collaboration rule* that steers/constrains the relation between the *location part* and *temporal part* of a *business collaboration behavior*.

Material Rule: a *business collaboration rule* that steers/constrains the *material part* of a *business collaboration behavior*.

Material-Functional Rule: a *business collaboration rule* that steers/constrains the relation between the *material part* and *functional part* of a *business collaboration behavior*.

Material-Location Rule: a *business collaboration rule* that steers/constrains the relation between the *material part* and *location part* of a *business collaboration behavior*.

Material-Participation Rule: a *business collaboration rule* that steers/constrains the relation between the *material part* and *participation part* of a *business collaboration behavior*.

Material-Temporal Rule: a *business collaboration rule* that steers/constrains the relation between the *material part* and *temporal part* of a *business collaboration behavior*.

Modality: refers to the usage of modals employed to qualify the truth of a judgement. In the *Business Collaboration Rule Language* each *business collaboration rule* may have modalities in its *antecedent* and/or *consequent*.

Model Schema: the collection of *elements* and their *policies* that together form the building blocks for an individual *model* in a *design* of a *business collaboration*.

Non-monoticity: represents the kind of inferencing in which reasoners reserve the right to retract conclusions in the light of new information.

Operational-Service Rule: a *business collaboration rule* that steers/constrains the *mapping* between a *business collaboration behavior* at *operational level* and *service level*.

Participation Rule: a *business collaboration rule* that steers/constrains the *participation part* of a *business collaboration behavior*.

Participation-Location Rule: a *business collaboration rule* that steers/constrains the relation between the *participation part* and *location part* of a *business collaboration behavior*.

Participation-Temporal Rule: a *business collaboration rule* that steers/constrains the relation between the *participation part* and *temporal part* of a *business collaboration behavior*.

Policy: a set of *policy alternatives* that describe possible courses of action that an organization may pursue. Concretely a policy is associated with an individual *element* governing how this *element* may be combined with others in a *design*.

Policy Alternative: a group of logically related *business collaboration rules* such that they steer/constrain (some part of) the business of the organization in a coherent, consistent and meaningful manner.

Prioritization: a mechanism for determining which *rules* may be overridden by other, higher-priority rules.

Process-Protocol Rule: a *business collaboration rule* that steers/constrains the mapping between a *business process* and *business protocol* at a particular *level*.

Promise: a *business collaboration rule* that steers/constrains the *participant public behavior aspect* in a *business collaboration*, i.e. a *rule* governing the behavior which an organization exposes to others.

Protocol-Agreement Rule: a *business collaboration rule* that steers/constrains the mapping between a *business protocol* and *business agreement* at a particular *level*.

Regulation: a *business collaboration rule* that steers/constrains the *internal business process aspect* in a *business collaboration*, i.e. a *rule* to which an organization internally adheres.

Rule: an accepted principle or instruction that states the way things are or should be done, and tells you what you are allowed or are not allowed to do. A rule consists of a *consequent* and (optional) *antecedent*.

Stipulation: a *business collaboration rule* that steers/constrains the *conversation aspect* in

a *business collaboration*, i.e. a *rule* which the involved organizations have agreed upon to follow.

Strategic-Operational Rule: a *business collaboration rule* that steers/constrains the *mapping* between a *business collaboration behavior* at *strategic level* and *operational level*.

Temporal Rule: a *business collaboration rule* that steers/constrains the *temporal part* of a *business collaboration behavior*.

Term: a noun or noun phrase with an agreed upon definition. In rule based business collaboration the *modeling description atoms* are the terms.

Chapter 6

Ambivalence: a type of rule anomaly which conveys that the application of a *rule* leads to an impermissible conclusion in a *business collaboration design*.

Business Collaboration Design Algorithm: the generic algorithm with which *business collaboration designs* are derived by combining *modeling description atoms* using their associated *derivation rules* whilst ensuring their *conformance*, *validity*, *alignment* and *compatibility* via their *control rules*.

Business Collaboration Management Algorithm: the generic algorithm with which *business collaboration designs* are managed when their *derivation rules* or *control rules* are modified whilst maintaining their *conformance*, *validity*, *alignment* and *compatibility*.

Circularity: a type of rule anomaly which conveys that one or more *rules* together cause a loop when applied that continues indefinitely. Considered to be harmful when negation and/or arithmetic operations are involved.

Conformance: represents the notion that *business collaboration designs* are compliant with their associated *business collaboration rules*. In context of *model theory* it implies that *modeling description atoms* are conform their *business collaboration rules* if they can be said to model these rules in the sense of *model theoretic truth*. Conformance is formally defined in terms of *Herbrand Universes* and *Herbrand Bases*.

Deficiency: a type of rule anomaly which conveys that there is a permissible *business collaboration design* such that it contains a fact that can not be deduced from the *rules*.

Redundancy: a type of rule anomaly which conveys that a *rule* is redundant, which is the case when if for every possible interpretation of the *rule*, i.e. in every possible *business collaboration design*, it does not matter whether it is applied or not.

Rule Based Business Collaboration System: a system that facilitates the rule based development and management of *business collaborations*.

Semantics Of Logic: approaches that logicians have introduced to understand and determine that part of meaning in which they are interested. An often used approach is that of *model theory*.

Well-Defined Design: a design defined in such a manner that is in compliance with the *derivation rules* specified in its *design schema* whilst at the same time adhering to the *control rules* in this schema.

Prototype Terms

Backward Chaining: the process in which a *rule engine* starts with one or more facts that it then attempts to prove by searching the available *rules* until it finds one whose then clause matches the desired fact. It then attempts to prove the *antecedent* of this *rule*.

Flow Based Rule Processing: the process in which a *rule engine* applies rules one at a time in a pre-defined order, sometimes also referred to as workflow based rule processing.

Forward Chaining: the process in which a *rule engine* starts with the facts available in the working memory and attempts to derive new facts through *inferencing*.

Icarus: the *prototype* implementation of a *rule based business collaboration system* conform the proposed approach.

Inferencing: the process of applying *rules* as performed by a *rule engine*. Inferencing may be done using *forward chaining* or *backward chaining*, or via *flow based rule processing*.

Prototype: a working model created via prototyping of a design in order to test various aspects, illustrate ideas or features and gather early user feedback.

Rule Engine: a software application that contains definitions of *rules*, and controls their selection and activation to enable *inferencing*. A rule engine is typically production system or logic system and may support *forward chaining*, *backward chaining* and *flow based rule processing*.

Bibliography

Aiello, M., Papazoglou, M., Yang, J., Carman, M., Pistore, M., Serafini, L., and Traverso, P. (2002). A Request Language for Web-services based on Planning and Constraint Satisfaction. In *Proceedings of the VLDB workshop on Technologies for E-Services*.

Aiken, A., Widom, J., and Hellerstein, J. (1995). Static Analysis Techniques for Predicting the Behavior of Active Database Rules. *Journal of ACM Transactions on Database Systems*, 20(1):3–41.

Akkermans, H., Gordijn, J., and van Vliet, H. (2000). Business Modelling is not Process Modelling. In *Proceedings of the International Conference on Conceptual Modeling for e-Business and the Web*.

Anderson, A. (2004). An Introduction to the Web Services Policy Language. In *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks*.

Andrieux, A., Czajkowski, K., and Dan, A. (2004). Web Services Agreement Specification (WS-Agreement).

Antonelli, G. (2006). Non-monotonic Logic.

Antoniou, G., Bikakis, A., and Wagner, G. (2004). A System for Nonmonotonic Rules on the Web. In *Proceedings of the RuleML 2004 conference*.

Apt, K. (2003). *Principles of Constraint Programming*. Cambridge University Press.

Arbab, F. (2004). Reo: a Channel-based Coordination Model for Component Composition. *Mathematical Structures in Computer Science*, 14(3).

Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., and Zimek, S. (2002). Web Service Choreography Interface.

Atkinson, B., Della-Libera, G., Hada, S., Hondo, M., Hallam-Baker, P., Klein, J., LaMacchia, B., Leach, P., Manferdelli, J., Maruyama, H., Nadalin, A., Nagaratnam, N., Prafullchandra, H., Shewchuk, J., and Simon, D. (2002). Web Services Security.

Atzeni, P., Ceri, S., Paraboschi, S., and Torlone, R. (1999). *Database Systems: Concepts, Languages and Architectures*. McGraw-Hill.

Bailey, J., Poulovassilis, A., and Wood, P. (2002). An Event-Condition-Action Language for XML. In *Proceedings of the 11th International Conference on the World Wide Web*.

Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., Nagaratnam, N., Prafullchandra, H., von Riegen, C., Roth, D., Schlimmer, J., Sharp, C., Shewchuk, J., Vedamuthu, A., Yalynalp, U., and Orchard, D. (2006). Web Services Policy Framework (WS-Policy).

Ball, M. (2006). OO jDREW Rule Engine.

Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., and Williams, S. (2004). Web Service Conversation Language (WSCL).

Baralis, E., Ceri, S., and Paraboschi, S. (1998). Compile-time and Runtime Analysis of Active Behaviors. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):353–370.

Basten, A. (1998). *In Terms of Nets, System Design with Petri Nets and Process Algebra*. PhD thesis, Department of Mathematics and Computing Science.

Bergstra, J. and Klop, J. (1985). Algebra of Communicating Processes With Abstraction. *Theoretical Computer Science*, 37(1):77–121.

Bergstra, J., Ponse, A., and Smolka, S. (2001). *Handbook of Process Algebra*. Elsevier.

Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005). Semantic Web Services Language (SWSL).

Booch, G., Rumbaugh, J., and Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web Services Architecture.

Bort, R. and Biefeldt, G. (1994). *Handbook of EDI*. Warren Gorham & Lamont.

Bowers, J., Button, G., and Sharrock, W. (1995). Workflow from Within and Without. In *Proceedings of the 4th European Conference on Computer-Supported Cooperative Work*.

Brambilla, M., Ceri, S., Comai, S., and Tziviskou, C. (2005). Exception Handling in Workflow-Driven Web Applications. In *Proceedings of WWW2005 International Conference*.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2004). Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Brinkkemper, S. (1995). Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology*, 38(4):275–280.

Brownston, L. (1985). *Programming Expert Systems in OPS5: an Introduction to Rule-Based Programming*. Addison-Wesley.

Bull, A. and Segerberg, K. (1984). *Basic Modal Logic*. Springer.

Business Process Modeling Initiative (2002). Business Process Modeling Language.

Business Process Modeling Initiative (2003). Business Process Modeling Notation.

Bussler, C. (2001). The Role of B2B Protocols in Inter-Enterprise Process Execution. In *Proceedings of the 2nd VLDB-Technologies for E-Services Workshop*.

Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., and Thatte, S. (2002). Web Services Transaction.

Cambridge Learner's Dictionary (2006).

Cardoso, J. and Sheth, A. (2003). Semantic E-Workflow Composition. *Journal of Intelligent Information Systems*, 21(3):191–225.

Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., and Shan, M. (2000). Adaptive and Dynamic Service Composition in eFlow. Technical Report HPL-2000-39, HP Lab.

Casati, F., Shan, E., Dayal, U., and Shan, M. (2003). Business-Oriented Management of Web Services. *Communications of the ACM*, 46(10):55–60.

Casati, F. and Shan, M. (2001). Dynamic and Adaptive Composition of E-Services. *Information Systems*, 6(3):143–163.

Chen, Q. and Hsu, M. (2001). Inter-Enterprise Collaborative Business Process Management. In *Proceedings of The International Conference on Data Engineering*.

Chisholm, M. (2004). *How to Build a Rule Engine*. Morgan Kaufmann Publishers.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Service Description Language.

Christophides, V., Hull, R., Kumar, A., and Simeon, J. (2000). Workflow Mediation using VorteXML. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(1):40–45.

Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002). Business Process Execution Language for Web Services.

Curtis, B., Kellner, M., and Over, J. (1992). Process Modeling. *Communications of the ACM*, 35(9):75–90.

DAML Services Coalition (2003). DAML-S.

Date, C. (2000). *What Not How: The Business Rule Approach to Application Development*. Addison-Wesley Longman Inc.

Dayal, U., Hsu, M., and Ladin, R. (2001). Business Process Coordination: State of the Art, Trends, and Open Issues. In *Proceedings of the 27th VLDB Conference*.

Decker, H. (1987). The Range Form or How to Avoid Floundering. Technical Report KB-26, ECRC.

Deiters, W., Goesmann, T., and Löffeler, T. (2000). Flexibility in Workflow Management: Dimensions and Solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):303–313.

Della-Libera, G., Dixon, B., Garg, P., Hada, S., Hallam-Baker, P., Hondo, M., Maruyama, H., Nagaratnam, N., Nash, A., Philpott, R., Prafullchandra, H., Shewchuk, J., Simon, D., Waingold, E., and Zolfonoon, R. (2005a). Web Services Secure Conversation Language (WS-SecureConversation).

Della-Libera, G., Hallam-Baker, P., Hondo, M., Janczuk, T., Kaler, C., Maruyama, H., Nagaratnam, N., Nash, A., Philpott, R., Prafullchandra, H., Shewchuk, J., Waingold, E., and Zolfonoon, R. (2005b). Web Services Security Policy (WS-SecurityPolicy).

DeMarco, T. (1978). *Structured Analysis and Systems Specifications*. Yourdon Inc.

d'Hondt, M. (2005). *Hybrid Aspects for Integrating Rule-Based Knowledge and Object-Oriented Functionality*. PhD thesis, System and Software Engineering Lab.

Dietrich, J. and Paschke, A. (2005). On the Test-Driven Development and Validation of Business Rules. In *Proceedings of the 4th International Conference on Information Systems Technology and its Applications*.

Dietz, J. (2006). *Enterprise Ontology: Theory and Methodology*. Springer.

Dijkman, R. and Dumas, M. (2004). Service-Oriented Design: A Multi-Viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368.

Dubray, J. (2003). A Novel Approach for Modeling Business Process Definitions.

Dybjer, P. (1997). *Semantics and Logics of Computation*. Cambridge University Press.

E. Mayol, E. T. (1995). Towards an Efficient Method for Updating Consistent Deductive Databases. In *Proceedings of the Basque International Workshop on Information Technology: Data Management Systems*.

ebXML Initiative (2002). Collaboration Protocol Profile and Agreement Specification.

ebXML Initiative (2006). Business Process Specification Schema.

Eder, J. and Liebhart, W. (1996). Workflow Recovery. In *Proceedings of the 1st International Conference on Cooperative Information Systems*.

Ellis, C. and Rozenberg, G. (1995). Dynamic Change within Workflow Systems. In *Proceedings of the Conference on Organizational Computing Systems*.

Fair Isaac (2006). Blaze Advisor.

Fensel, D. and Bussler, C. (2002). The Web Service Modeling Framework (WSMF). *Electronic Commerce Research and Applications*, 1(2):113–137.

Fitting, M. (2002). Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1-2):25–51.

Flach, P. (1994). *Simply Logical: Intelligent Reasoning by Example*. John Wiley & Sons Ltd.

Forgy, C. (1982). RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37.

Froland, S. and Koistinen, J. (1998). Quality-of-Service Specification in Distributed Object Systems. *Distributed System Engineering Journal*, 5(4):179–202.

Galton, A. (2006). Temporal Logic.

Garson, J. (2006). Modal Logic.

Gelfond, M. and Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *Proceedings of the Fifth International Conference on Logic Programming*.

Georgakopoulos, D., Hornick, M., and Sheth, A. (1995). An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. *Distributed and parallel databases*, 3(2):119–152.

Georgakopoulos, D., Schuster, H., Baker, D., and Cichocki, A. (2000). Managing Escalation of Collaboration Processes in Crisis Mitigation Situations. In *Proceedings of the 16th International Conference on Data Engineering*.

Geppert, A. and Tombros, D. (1998). Event-Based Distributed Workflow Execution with EVE. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*.

Ginsberg, A. (1988). Knowledge Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy. In *Proceedings of the 7th National Conference on Artificial Intelligence*.

Gordijn, J., Yu, E., and van der Raadt, B. (2006). E-Service Design Using i∗ and e3value Modeling. *IEEE Software*, 23(3):26–33.

Grefen, P., Aberer, K., Hoffner, Y., and Ludwig, H. (2000). Crossflow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises. *International Journal of Computer Systems Science & Engineering*, 15(5):277–290.

Grosof, B. (2004). Representing E-Commerce Rules Via Situated Courteous Logic Programs in RuleML. *International Journal of Electronic Commerce*, 8(4):61–97.

Grosof, B., Labrou, Y., and Chan, H. (1999). A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In *Proceedings of the 1st ACM Conference on Electronic Commerce*.

Hagen, C. and Alonso, G. (2000). Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering*, 26(10):943–958.

Haley Inc (2006). HaleyAuthority Edge.

Hamadi, R. and Benatallah, B. (2003). A Petri Net-Based Model for Web Service Composition. In *Proceedings of the 14th Australasian Database Conference*.

Han, Y., Sheth, A., and Bussler, C. (1998). A Taxonomy of Adaptive Workflow Management. In *Proceedings of the CSCW Workshop Towards Adaptive Workflow Systems*.

Hatley, D., Hruschka, P., and Pirbhai, I. (2000). *Process for System Architecture and Requirements Engineering*. Dorset House Publishing.

Hewitt, C., Bishop, P., and Steiger, R. (1973). A Universal Modular Actor Formalism for Artificial Intelligence. In *Proceedings of the third International Conference On Artificial Intelligence*.

Hoare, C. (1985). *Communicating Sequential Processes*. Prentice Hall.

Hodges, W. (2006). Model Theory.

Holzmann, G. (2003). *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2003). SWRL: A Semantic Web Rule Language Combining OWL and RuleML.

Huff, B., Presley, A., and Whitman, L. (1998). Issues Encountered between Model Views. In *Proceedings of the Conference on Flexible Automation and Integrated Manufacturing Conference Proceedings*.

IBM (2002). IBM CommonRules.

IBM (2006). IBM Service-Oriented Modeling and Architecture (under development).

ILOG (2006). ILOG JRules.

Jennings, N., Faratin, P., Norman, T., O'Brien, P., and Odgers, B. (1996). Autonomous Agents for Business Process Management. *International Journal of Applied Artificial Intelligence*, 14(2):145–189.

Joeris, G. and Herzog, O. (1999). Towards Flexible and High Level Modeling and Enacting of Processes. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering*.

Jonkers, H., Lankhorst, M., van Buuren, R., Hoppenbrouwers, S., and Bonsangue, M. (2003). Concepts for Modelling Enterprise Architectures. In *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference*.

JSR94 (2006). Jsr94.

Kimberley, P. (1991). *Electronic Data Interchange*. McGraw Hill.

Laukkanen, M. and Helin, H. (2003). Composing Workflows of Semantic Web Services. In *Proceedings of the AMAS Workshop on Web Services and Agent-Based Engineering*.

Laurent, J. and Ayel, M. (1989). Off Line Coherence Checking for Knowledge Based Systems. In *Proceedings of the IJCAI Workshop on Verification And Validation and Testing of Knowledge Based Systems*.

Leblanc, H. (1976). *Truth-Value Semantics*. Elsevier.

Leemans, N., Treur, J., and Willems, M. (2002). A Semantical Perspective on Verification of Knowledge. *Journal Of Data and Knowledge Engineering*, 40(1):33–70.

Leune, K., Papazoglou, M., and van den Heuvel, W. (2004). Specification and Querying Security Constraints in the EFSOC Framework. In *Proceedings of the 2d International Conference on Service Oriented Computing*.

Li, J., Mai, W., and Butler, G. (2003). Implementing Exception Handling Policies for Workflow Management System. In *Proceedings of the 10th Asia-Pacific Software Engineering Conference*.

Liles, D. and Presley, A. (1996). Enterprise Modeling Within An Enterprise Engineering Framework. In *Proceedings of the Winter Simulation Conference*.

Little, M., Newcomer, E., and Pavlik, G. (2005). Web Services Coordination Framework Specification (WS-CF).

Liu, C., Orlowska, M., Lin, X., and Zhou, X. (2001). Improving Backward Recovery in Workflow Systems. In *Proceedings of the 7th International Conference on Database Systems for Advanced Applications*.

Liu, L. and Pu, C. (1997). ActivityFlow: Towards Incremental Specification and Flexible Coordination of Workflow Activities. In *Proceedings of the 16th International Conference on Conceptual Modeling*.

Liu, Y. (2001). *Rule Warehouse System For Knowledge Sharing and Business Collaboration*. PhD thesis, Department of Computer and Information Science and Engineering.

Lloyd, J. and Topor, R. (1984). Making prolog more expressive. *Journal of Logic Programming*, 1(3):225–240.

Lloyd, J. and Topor, R. (1986). A Basis for Deductive Database Systems. *Journal of Logic Programming*, 13(1):55–67.

Lokhorst, G. (2006). Mally's Deontic Logic.

Loos, P. and Fettke, P. (2001). Towards an Integration of Business Process Modeling and Object-Oriented Software Development. In *Proceeding of the Fifth International Symposium on Economic Informatics*.

Ludwig, H., Keller, A., Dan, A., King, R., and Franck, R. (2003). Web Service Level Agreement Version 1.0.

Mandarax Project (2006). Mandarax.

Manola, F. and Miller, E. (2004). RDF Primer.

Marjanovic, O. and Milosevic, Z. (2001). Towards Formal Modeling of e-Contracts. In *Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference*.

Marshak, R. (1994). Software to Support BPR - The Value of Capturing Process Definitions. *Workgroup Computing Report*, 17(7).

McGovern, J., Tyagi, S., Stevens, M., and Mathew, S. (2003). *Java Web Services Architecture*. Morgan Kaufmann Publishers.

McGuinness, D. and van Harmelen, F. (2004). OWL Web Ontology Language Overview.

McIlraith, S. and Son, T. (2002). Adapting Golog for Composition of Semantic Web Services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning*.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. K. (2003). Composing Web Services on the Semantic Web. *VLDB Journal*, 12(4):333–351.

Mendling, J. and Nttgens, M. (2004). Exchanging EPC Business Process Models with EPML. In *Proceedings of the 1st GI Workshop XML Interchange Formats for Business Process Management*.

Meng, J., Su, S. Y., Lam, H., and Helal, A. (2002). Achieving Dynamic Inter-Organizational Workflow Management by Integrating Business Processes, Events, and Rules. In *Proceedings of the 35th Hawaii International Conference on System Sciences*.

Mentzas, G., Halaris, C., and Kavadias, S. (2001). Modelling Business Processes with Workflow Systems: an Evaluation of Alternative Approaches. *International Journal of Management*, 12(4):123–135.

Meyer, B. (2000). *Object Oriented Software Construction*. Prentice Hall.

Milner, R. (1990). *Operational and Algebraic Semantics of Concurrent Processes*. Elsevier.

Milner, R. (1993). *Operational and Algebraic Semantics of Concurrent Processes*. Elsevier.

Moriarty, T. (1993). The Next Paradigm. *Database Programming and Design*, 6(2):66–69.

Nadalin, A., Kaler, C., Monzillo, R., and Hallam-Baker, P. (2006). Web Service Security.

Nagy, A., Fairchild, A., and Orriëns, B. (2004). The Promise and Reality of Internet-Based Interorganizational Systems. In *Proceedings of the E-Society IADIS International Conference*.

Narayanan, S. and McIlraith, S. (2003). Analysis and Simulation of Web Services. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 42(5):675–693.

Nezhad, H., Benatallah, B., Casati, F., and Toumani, F. (2006). Web Service Interoperability Specifications Explained and Analyzed. *IEEE Computer*, 39(5):24–32.

Nguyen, G. and Vernadat, F. (1994). Cooperative Information Systems in Integrated Manufacturing Environments. In *Proceedings of the 2d International Conference on Cooperative Information Systems*.

Nolan, P. (2004). Understand WS-Policy Processing.

Nuseibeh, B., Kramer, J., and Finkelstein, A. (2003). Viewpoints: meaningful relationships are difficult! In *Proceedings of the 25th International Conference on Software Engineering*.

OASIS Legal XML eContracts Technical Committee (2006).

OASIS Semantic Execution Environment Technical Committee (2006). Web Services Modeling Language.

Object Modeling Group (2003a). MDA Guide 1.1.

Object Modeling Group (2003b). Object Constraint Language.

Object Modeling Group (2006). Semantics of Business Vocabulary and Business Rules Specification.

OBrien, P. and Wiegand, W. (1998). Agent based Process Management: Applying Intelligent Agents to Workflow. *The Knowledge Engineering Review*, 13(2):1–14.

Orriëns, B. (2006a). Specification of Assessment Requirements for Business Collaborations. Technical Report 28, Infolab Technical Report Series.

Orriëns, B. (2006b). Specification of Payment Requirements for Business Collaborations. Technical Report 27, Infolab Technical Report Series.

Orriëns, B. (2006c). Specification of Quality Requirements for Business Collaborations. Technical Report 26, Infolab Technical Report Series.

Orriëns, B. (2006d). Specification of Security Requirements for Business Collaborations. Technical Report 25, Infolab Technical Report Series.

Orriëns, B. (2007). Dissertation home.

Orriëns, B. and Yang, J. (2005). Bridging the Gap between Business and IT in Service Oriented Business Collaboration. In *Proceedings of the IEEE International Conference on Services Computing*.

Orriëns, B., Yang, J., and Papazoglou, M. (2005). A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaboration. In *Proceedings of the 3d International Conference on Service Oriented Computing*.

Papazoglou, M. (2003). Web Services and Business Transactions. *World Wide Web: Internet and Web Information Systems*, 6(1):49–91.

Papazoglou, M. (2006). Web Services Technologies and Standards. *ACM Computing Surveys (to be published)*.

Papazoglou, M., Aiello, M., Pistore, M., and Yang, J. (2002). XSRL: An XML Web-Service Request Language. Technical Report DIT-02-0079, University of Trento.

Papazoglou, M. and Georgakopoulos, D. (2003). Service-Oriented Computing. *Communications of the ACM*, 46(10):25–28.

Parnas, D. (1972). On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058.

Paschke, A. (2005). RBSLA - A Declarative Rule-Based Service Level Agreement Language based on RuleML. In *Proceedings of the International Conference on Intelligent Agents*.

Peltz, C. (2003). Web Services Orchestration: a Review of Emerging Technologies, Tools, and Standards.

Phalpa, K., Henderson, P., Walters, R., and Abeysinghec, G. (1998). RolEnact: Role-Based Enactable Models of Business Processes. *Journal of Information and Software Technology*, 3(40):123–133.

Preece, A., Shinghal, R., and Batarekh, A. (1992). Principles and Practice in Verifying Rule Based Systems. *The Knowledge Engineering Review*, 7(2):115–141.

Radhakrishna, P., Karlapalem, K., and Dani, A. (2005). From Contracts to E-Contracts: Modeling and Enactment. *Information and Technology Management*, 6(4):363–387.

Rao, J., Kungas, P., and Matskin, M. (2006). Composition of Semantic Web Services using Linear Logic Theorem Proving. *International Journal Of Information systems*, 31(4-5):340–360.

Rao, J. and Su, X. (2004). A Survey of Automated Web Service Composition Methods. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*.

Reichert, M. and Dadam, P. (1997). A Framework for Dynamic Changes in Workflow Management Systems. In *Proceedings of the 8th International Conference on Database and Expert Systems Applications*.

Reichert, M. and Dadam, P. (1998). ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129.

Rittgen, P. (2000). EMC- A Modeling Method for Developing Web-Based Applications. In *Proceedings of the International Conference of the International Resources Management Association*.

Rosenberg, F. and Dustdar, S. (2005). Business Rules Integration in BPEL A Service-Oriented Approach. In *Proceedings of the 7th International IEEE Conference on E-Commerce Technology*.

RosettaNet (2006).

Ross, R. (1997). *The Business Rule Book: Classifying and Modeling Rules*. Business Rule Solutions.

Ross, R. (2003). *Principles of the Business Rule Approach*. Addison-Wesley.

RuleML Initiative (2006). Rule Markup Language.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence A Modern Approach*. Prentice Hall.

Sadiq, W. and Orlowska, M. (2000). On Business Process Model Transformations. In *Proceedings of 19th International Conference on ER*.

Sandia National Laboratories (2006). JESS.

Scheer, A. (1992). *Architecture of Integrated Information Systems: Foundations of Enterprise Modelling*. Springer.

Schreiber, A., Akkermans, J., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W. V., and Wielinga, B. (1999). *Knowledge Engineering and Management: The CommonKADS Methodology*. MIT Press.

ServiceMosaic (2006). Servicemosaic.

Shankar, R., Ponnekanti, R., and Fox, A. (2002). Sword: A Developer Toolkit for Web Service Composition. In *Proceedings Of The Eleventh World Wide Web Conference*.

Sheng, Z., Benatallah, B., Dumas, M., and Mak, E. (2002). SELFSERV A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In *Proceedings of the 28th Very Large Data Bases Conference*.

Stevens, M. (2003). Service Oriented Architecture.

Tartanoglu, F., Issarny, V., Romanovsky, A., and Levy, N. (2003). Coordinated Forward Error Recovery for Composite Web Services. In *Proceedings of the 22nd IEEE Symposium on Reliable Distributed Systems*.

Topor, R. (1991). Safe database queries with arithmetic relations. In *Proceedings of the 14th Australian Computer Science Conference*, pages 1–13.

Tosic, V., Patel, K., and Pagurek, B. (2003). WSOL - Web Service Offerings Language. In *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web*.

Traverso, P., Pistore, M., Roveri, M., Marconi, A., Kazhamiakin, R., Lucchese, P., Busetta, P., and Bertoli, P. (2004). Supporting the Negotiation between Global and Local Business Requirements in Service Oriented Development. In *Proceedings of the 2d International Conference on Service Oriented Computing*.

Ullman, J. and Widom, J. (1997). *A First Course in Database Systems*. Addison-Wesley.

van den Heuvel, W. (2002). *Integrating Modern Business Applications with Objectified Legacy Systems*. PhD thesis, Department Of Information Management.

van der Aalst, W. (1998). The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66.

van der Aalst, W., Basten, T., Verbeek, H., Verkoulen, P., and Voorhoeve, M. (1999a). Adaptive Workflow - An Approach Based on Inheritance. In *Proceedings of the IJCAI Workshop on Intelligent Workflow and Process Management*.

van der Aalst, W., Basten, T., Verbeek, H., and Voorhoeve, M. (1999b). Adaptive Workflow - On the Interplay between Flexibility and Support. In *Proceedings of the International Conference on Enterprise Information Systems*.

van der Aalst, W., ter Hofstede, A., and Weske, M. (2003). On the Application of Formal Methods to Process-Aware Information Systems. In *Proceedings of the International Conference on Business Process Management*.

van Gelder, A., Ross, K., and Schlipf, J. (1991). The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650.

Verbeek, E. and van der Aalst, W. (2000). Woflan 2.0 - A Petri-Net-Based Workflow Diagnosis Tool. In *Proceedings of the 21st International Application and Theory of Petri Nets*.

Vernadat, F. (1992). *CIMOSA - A European Development for Enterprise Integration (Part 2): Enterprise Modeling*. Pergamon Press Inc.

Veryard, R. (2002). Rule Based Development. *CBDi Journal*.

Veryard, R. (2003). Modeling for SOA. *CBDi Journal*.

von Halle, B. (2002). *Business Rules Applied: Building Better Systems Using the Business Rule Approach*. John Wiley & Sons Ltd.

von Halle, B. and Sandifer, A. (1991). Designing by the Rules. *Database Programming and Design*, 4(1):11–14.

Wagner, F. (2006). *Modeling Software with Finite State Machines: A Practical Approach*. Auerbach Publications.

Wagner, G. (2002). How to Design a General Rule Markup Language? In *Proceedings of the Workshop XML Technologies for the Semantic Web*.

Wagner, G. (2003). The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior. *Information Systems*, 28(5):475–504.

Weigand, H. (2004). Research Methods and Methodology.

Winograd, T. and Flores, R. (1987). *Understanding Computers and Cognition: a New Foundation for Design*. Addison-Wesley.

Workflow Management Coalition (1995). The Workflow Reference Model.

Workflow Management Coalition (2002). Workflow Process Definition Interface – XML Process Definition Language.

Wu, P. (1993). *Rule Validation in Object-oriented Knowledge Base*. PhD thesis, Department of Electrical Engineering.

Xu, L. (2004). *Monitoring Multi-Party Contracts for E-Business*. PhD thesis, Department of Information Management.

Yang, J. and Papazoglou, M. (2002). Web Component: A Substrate for Web Service Reuse and Composition. In *Proceedings of Advanced Information Systems Engineering 14th International Conference*.

Yasu Technologies (2006). QuickRules 4.0.

Yourdon, E. (1988). *Modern Structured Analysis*. Prentice Hall.

Yu, E. (1997). Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*.

Zachman, J. (1987). A Framework for Information Systems Architecture. *IBM Systems Journal*, 26(3):276–292.

Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer, D., and Chang, H. (2003). Flexible Composition of Enterprise Web Services. *Electronic Markets - The International Journal of Electronic Commerce and Business Media*, 13(2):141–152.

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. (2004). QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327.

Samenvatting

Dat wil ik graag laten weten: deze feiten bevinden zich in de samenvatting welke mijns inziens een erg goede is; Vrij vertaald van John Sherman Cooper

Het schrijven van lange boeken is een arbeidsintensieve en verarmende activiteit van stompzinnigheid: het uitbreiden van een idee tot driehonderd pagina's dat perfect in een paar minuten uitgelegd kan worden. Een betere manier is om te doen alsof deze boeken al bestaan en een samenvatting aan te bieden: Vrij vertaald van Jorge Luis Borges

Het huidige bedrijfsklimaat wordt gekenmerkt door een hoge mate van veranderlijkheid wat grote gevolgen heeft voor organisaties die zwaar leunen op hun IT-infrastructuur. Zulke organisaties moeten het hoofd bieden aan zich snel wijzigende marktsituaties, nieuwe concurrenten, veranderende wettelijke verplichtingen, en opkomende competitieve dreigingen. Deze en andere trends zorgen ervoor dat organisaties genoodzaakt zijn om hun IT-infrastructuur snel aan te passen om nieuwe bedrijfsmodellen en vereisten te kunnen ondersteunen. Alleen op deze wijze kan een organisatie zich staande houden in een wereld waarin samenwerking met andere organisaties grotendeels plaatsvindt via semi-geautomatiseerde en complexe electronische transacties. Het bouwen en onderhouden van dergelijke bedrijfssamenwerkingen die de grenzen van organisaties en hun IT systemen overschreiden is een uitdaging aangezien het vereist dat de componenten van de individuele organisaties aan elkaar moeten worden gelinkt opdat ze een coherent geheel vormen. De kunst is om dit te realiseren op een zodanige wijze dat de eisen die aan iedere organisatie in de samenwerking worden gesteld haalbaar zijn voor deze organisaties, terwijl tegelijkertijd de samenwerking als geheel consistent blijft. Het dynamische bedrijfsklimaat draagt er aan bij dat deze uitdaging alleen nog maar groter wordt.

Een technologie welke wellicht zulke dynamische bedrijfssamenwerkingen binnen bereik kan brengen is dat van 'Service Oriented Computing' (SOC) gebaseerde 'middleware'. SOC maakt het mogelijk om op een standaard wijze heterogene systemen en applicaties (mogelijk behorende tot verschillende organisaties) te integreren die hun diensten aanbieden als 'services'. Een typische ontwikkeling van bedrijfssamenwerkingen binnen SOC is om de bedrijfsprocessen van een organisatie te omschrijven, vervolgens protocollen op te stellen die vastleggen hoe de organisatie wenst samen te werken met andere organisaties, en tenslotte overeenkomsten te sluiten die definiëren op welke wijze samenwerking dient plaats te vinden. Huidige oplossingen voor de realizatie hiervan zijn ontwikkeld zowel bin-

nen de software industrie alsmede de academische wereld. Echter, deze oplossingen zijn veelal op techniek gefocused en vereisen vaak het handmatig ontwikkelen en onderhouden van de modellen. Daardoor verliezen zij uit het oog dat technologie juist secundair is aan het beleid, bedrijfsregels, informatie en processen die gebruik maken van de aangeboden services. Dit maakt het lastig om ervoor te zorgen dat deze services in overeenstemming zijn met de vereisten vanuit de organisatie. Tevens voldoet een handmatige aanpak niet om in het licht van snel veranderende vereisten in een dynamisch bedrijfsklimaat.

Om dit probleem te verhelpen wordt in het proefschift een alternatieve methode beschreven voor het dynamische ontwikkelen van bedrijfssamenwerkingen basis van SOC via het gebruik van regels. De voorgestelde aanpak is gestoeld op drie ideëen: ten eerste wordt het ontwikkelingsproces opgesplitst in meerdere delen om de complexiteit van samenwerkingen tussen organisaties te verminderen en op deze wijze de beheersbaarheid ervan te vergroten. Tegelijkertijd worden de afhankelijkheden tussen deze delen in acht genomen om ervoor te zorgen dat samenwerkingen als geheel consistent blijft. Ten tweede wordt een op modellen gebaseerde aanpak geintroduceerd waarmee de verschillende delen alsmede hun afhankelijkheden op een expliciete manier gespecificeerd kunnen woren. Deze modellen fungeren dan als de basis aan de hand waarvan daadwerkelijke samenwerkingen tussen organisaties tot stand kunnen komen alsmede voor de communicatie van vereisten voor dergelijke samenwerkingen tussen de betrokken organisaties. Ten derde wordt een op regels gebaseerd mechanism ontwikkeld om dergelijke samenwerkingsmodellen te maken op een zodanige wijze dat pas tijdens de daadwerkelijke uitvoering van de samenwerking modellen ervan ontwikkeld worden. Door het wijzigen van regels kunnen modellen van nieuwe samenwerkingen dan makkelijk gemaakt worden, terwijl bestaande modellen van samenwerkingen aangepast kunnen worden door het toevoegen van nieuwe of aanpassen van bestaande regels.

Author Index

Index

Curriculum Vitae

Resume: a written exaggeration of only the good things a person has done in the past, as well as a wish list of the qualities a person would like to have; Bo Bennett

I'm the guy to call. Look at the resume. I have kids of my own. I have dogs; Jeff Daniels

Bart Orriëns was born on November 4, 1978 in Doetinchem, The Netherlands. He finished secondary school at Isala College in Silvolde. He went on to study Information Management at Tilburg University in Tilburg. In 2001 he graduated with a Master's thesis on the requirements for e-business registries to facilitate the discovery of electronic services under supervision of Jian Yang. After his graduation Bart explored the field of Artificial Language (in particular language analysis and recognition) with a partial study on Language and Artificial Intelligence at Tilburg University. Concurrently Bart also worked for the Department of Information Management at Tilburg University where he did research on the implementation of a web service composition language developed at Tilburg University's INFOLAB. In March of 2003 he accepted a position as a PhD student at the same department as a continuation of this early research.

In his four years as a PhD student his interests lied in the area of business rules and composition of automated services with a specific interest to develop theories, techniques and methodologies that enable established organizations to expand their corporate IT systems beyond the boundaries of the traditional organizations and interoperate other systems in other organizations while guaranteeing consistency of business rules. As a result he acquired considerable expertise in the domains of Service Oriented Architectures, XML Web Services, service composition and orchestration as well as business process modeling, constraint analysis, rule driven software generating. Moreover, he became well skilled in the entirety of software development from the initial phase of requirements analysis to actual prototyping.

Results of his research were published in numerous conferences and journals for example the various SOC and SCC/ICWS conferences as well as the Journal Of Integrated Design and Process Science (June 2004, Vol. 8, No. 2) and the Special Issue of the International Journal on Business Process Management on Web Services (to be published). He also collaborated with many high profile individuals in his field of interest among others during his 1,5 year stay as a visiting scholar at Macquarie University in Sydney, Australia. In addition to this he also contributed to the scientific community by acting as a peer reviewing

articles. Bart was also involved in teaching activities. These consisted foremost of teaching both lectures and labs for the 3d year course of Object-Oriented Modelling. In addition, he also gave several guest lectures for such courses as Business Application Programming and E-Business. Moreover, he supervised several student bachelor and master projects.

SIKS Dissertation Series

Other books that have appeared in the SIKS Dissertation Series:

1998-1	Johan van den Akker (CWI) DEGAS - An Active, Temporal Database of Autonomous Objects
1998-2	Floris Wiesman (UM) Information Retrieval by Graphically Browsing Meta-Information
1998-3	Ans Steuten (TUD) A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
1998-4	Dennis Breuker (UM) Memory versus Search in Games
1998-5	E.W.Oskamp (RUL) Computerondersteuning bij Straftoemeting
1999-1	Mark Sloof (VU) Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
1999-2	Rob Potharst (EUR) Classification using decision trees and neural nets
1999-3	Don Beal (UM) The Nature of Minimax Search
1999-4	Jacques Penders (UM) The practical Art of Moving Physical Objects
1999-5	Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
1999-6	Niek J.E. Wijngaards (VU) Re-design of compositional systems
1999-7	David Spelt (UT) Verification support for object database design
1999-8	Jacques H.J. Lenting (UM)

Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6	Martijn van Welie (VU) Task-based User Interface Design
2001-7	Bastiaan Schonhage (VU) Diva: Architectural Perspectives on Information Visualization
2001-8	Pascal van Eck (VU) A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9	Pieter Jan 't Hoen (RUL) Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10	Maarten Sierhuis (UvA) Modeling and Simulating Work Practice, BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11	Tom M. van Engers (VUA) Knowledge Management: The Role of Mental Models in Business Systems Design
2002-01	Nico Lassing (VU) Architecture-Level Modifiability Analysis
2002-02	Roelof van Zwol (UT) Modelling and searching web-based document collections
2002-03	Henk Ernst Blok (UT) Database Optimization Aspects for Information Retrieval
2002-04	Juan Roberto Castelo Valdueza (UU) The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05	Radu Serban (VU) The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06	Laurens Mommers (UL) Applied legal epistemology; Building a knowledge-based ontology of the legal domain
2002-07	Peter Boncz (CWI) Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08	Jaap Gordijn (VU) Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09	Willem-Jan van den Heuvel(KUB)

Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (UvA)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)

Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politile gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))

AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)

Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)

Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)

Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems