Requirements-Driven Adaptation of Choreographed Interactions

Thesis

Requirements-Driven Adaptation of Choreographed Interactions

Ayman Mahfouz

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University
United Kingdom

2012

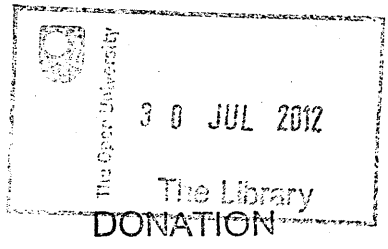
DATE OF SUBMISSION: 2 MARCH 2012
DATE OF AWARD: 21 JUNE 2012

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Abstract

Electronic services are emerging as the de-facto enabler of interaction interoperability across organization boundaries. Cross-organizational interactions are often "choreographed", i.e. specified by a messaging protocol from a global point of view independent of the local view of each interacting organization. Local requirements motivating an interaction as well as the global contextual requirements governing the interaction inevitably evolve over time, requiring adaptation of the corresponding interaction protocol. Adaptation of an interaction protocol must ensure the satisfaction of both sets of interaction requirements while maintaining consistency between the global view and the local views of an interaction specification. Such adaptation is not possible with the current state-of-the-art representations of choreographed interactions, as they capture only operational messaging specifications detached from both local organizational requirements as well as global contextual requirements.

This thesis presents three novel contributions that tackle adaptation of choreographed interaction protocols: an automated technique for deriving an interaction protocol from requirements, a formalization of consistency between local and global views, and a framework for guiding the adaptation of a choreographed interaction. A choreographed interaction is specified using models of organizational requirements motivating the interaction. We employ the formal semantics embedded in requirements models to automatically derive an interaction protocol. We propose a framework for relating the global and local views of interaction specification and maintaining consistency between them. We develop a metamodel for interaction specification, from which we enumerate adaptation operations. We build a catalogue that provides guidance on performing each operation and propagating changes between the global and local views. These contributions are evaluated using examples from the literature as well as a real-world case study.



Author's Declaration

All of the work presented in this thesis describes original contributions of the author and none of it has been submitted previously for any degree at any institution. Much of the material in this thesis appears in the following publications:

- Mahfouz, A., Barroca, L., Laney, R., & Nuseibeh, B. (2006). "Patterns for service-oriented information exchange requirements," in Proceedings of the 2006 Conference on Pattern Languages of Programs (PLOP'06). Portland, Oregon, USA, October 21-23, 2006: ACM.
- Mahfouz, A., Barroca, L., Laney, R., & Nuseibeh, B. (2008). "Customizing Choreography: Deriving Conversations from Organizational Dependencies," in *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC'08)*. Munich, Germany: IEEE Computer Society, 15-19 Sept, pp. 181-190.
- Mahfouz, A., Barroca, L., Laney, R., & Nuseibeh, B. (2009). "From Organizational Requirements to Service Choreography," in *Proceedings of the Third International Workshop of Software Engineering for Adaptive Service-oriented Systems (SEASS 2009) held in conjunction with the First International Conference on Cloud Computing (CLOUD-I 2009)*, Los Angeles, California, USA: IEEE Computer Society, 6-10 July, pp. 546-553.
- Mahfouz, A., Barroca, L., Laney, R., & Nuseibeh, B. (2009). "Requirements-Driven Collaborative Choreography Customization," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'09)*, vol. 5900. Stockholm, Sweden: Lecture Notes in Computer Science (Springer), 24-27 Nov, pp. 144-158
- Mahfouz, A. (2010). "Requirements-Driven Design of Service-Oriented Interactions: An Evaluation," The Open University, Milton Keynes, Technical Report TR2010/11, July.
- Mahfouz, A., Barroca, L., Laney, R., & Nuseibeh, B. (2010). "Requirements-Driven Design of Service-Oriented Interactions," IEEE Software no. 0740-7459/10 (November/December), pp. 25-32.

All publications were peer-reviewed except for the technical report, albeit it was submitted along with the IEEE Software article as supporting material.

Acknowledgements

I wish to thank my supervisors, Leonor Barroca, Robin Laney, and Bashar Nuseibeh for their invaluable support. The completion of this thesis could not have been possible without their guidance, thorough reviews, and continuous encouragement, especially during tough times.

I also want to thank my wife Ayat for her continuous support during the process and her understanding that carried me through the countless hours I spent on my research. Part of the thanks has to go to my sons Omar and Rasheed, their mere existence was a constant motivation to finish my thesis so that I can spend more time with them. A special thanks goes to my parents for their life-long support and encouragement to pursue a Doctoral degree.

Thanks are also due to fellow students and staff at The Department of Computing of The Open University for their support and inspiring discussions.

Finally, I would like to thank my fellow Egyptians who gave their lives in Tahrir Square and whose blood fueled my belief that anything is possible if the will exists.

Table of Contents

Table of Figures

Chapter 1. Introduction

During the past five decades, software engineering has undergone fundamental shifts in accepted wisdom both for products and process (Larman & Basili, 2003). On the product side, enterprise software applications evolved from being monolithic, standalone, closed systems to being distributed, heterogeneous, open-ended systems. Shared, rather than single, ownership of applications has become the norm rather than the exception (Nitto et al., 2008). On the process side, the assumption that requirements should be fixed in early stages of the development process was ousted by the belief that change in requirements at later stages is inevitable (Edmonds, 1974). Interoperability between heterogeneous systems and adaptability of software products emerged as two of the primary challenges in developing modern era enterprise software. Software development processes and supporting technologies needed to evolve to meet these challenges.

In the past decade, Service-oriented computing (SOC) (Georgakopoulos & Papazoglou, 2008) emerged as a promising approach for enabling cross-organizational interoperability. SOC promotes hiding the heterogeneity of diverse software systems behind service interfaces. Service interfaces define the structure and semantics of electronic messages that clients exchange with services to access business functions. Standards-based languages for describing service interfaces abstract the specifics of an organization's IT infrastructure, thereby enabling interoperability in distributed information systems (Alonso et al., 2004).

It is widely recognized that interaction between distributed components to perform a certain task is the most important characteristic of a distributed system (Wooldridge, 2002). For cross-organizational interactions, abstracting computational infrastructure and business functions is but a first step towards establishing interoperability. Enterprises may employ proprietary business processes that impose constraints on how their business is conducted. An organization's business process may require carrying out its business activities in a certain sequence or mandate that

certain data flows take place. The specifics of business processes employed by organizations wishing to interact may render their businesses mutually incompatible.

Not only do services provide interoperability at the interface level, but they also provide means for bridging incompatible business processes. A number of languages have been proposed (Papazoglou et al., 2006) for specifying standards-based cross-organizational interactions. Using these languages message exchange sequences expected to take place in the course of interaction between organizations can be specified, thereby establishing an interoperable interaction protocol. An interaction protocol can be published and made available to organizations wishing to interact. Organizations can thus assess interoperability of their businesses with reference to an interaction protocol and adapt their processes, if necessary, to be compatible with it.

Whereas each organization participating in an interaction is a stakeholder with a "local" view that embodies its goals, business processes, and operational constraints, cross-organizational interaction protocols are typically specified from a "global" point of view. The global view allows a neutral observer, such as a regulatory agency overseeing an interaction, to assess the adherence of participants to the interaction protocol. By definition, the global view is concerned with observable exchanges and abstracts away from the internals of business processes of interaction participants. Choreography Description Languages (CDLs) emerged as a means for specifying interaction protocols from the point of view of a neutral stakeholder (Austin et al., 2004) who wishes to monitor the observable behavior of interacting participants (i.e. the global view). An interaction described using a CDL is said to be "choreographed".

While choreographed interaction protocols deal with the interoperability challenge, they do not address the adaptability challenge. It is inevitable that the business needs driving an organization to participate in an interaction will change or get augmented with new needs. It is also likely that the context of an interaction, most notably the business regulations governing the interaction, will change. In both cases, the protocol of the interaction has to be adapted to satisfy the emerging business needs. A number of challenges have to be faced when adapting cross-organizational interaction protocols.

The first challenge is how to make decisions about possible alternative ways for adapting an interaction protocol. The difficulty here is mainly due to deficiencies in state-of-the-art CDLs. These languages typically describe operational aspects of an interaction such as control flow and messaging-sequence specification. The goals of the interacting organizations and their business needs are not directly represented in CDLs. It is thus hard to reason about whether an interaction protocol, or an adaptation of it, satisfies an organization's business needs. Furthermore, these languages describe only the electronic part of an interaction, whereas physical activities that are an integral part of it (e.g. receiving a shipment) are not specified in an interaction protocol. It is thus impossible to ensure that an adaptation of the protocol satisfies the business needs of the stakeholders by referring only to the CDL representation of an interaction, especially with the lack of formality associated with these languages (van der Aalst, 2003).

Secondly, whereas an interaction protocol is described from a global point of view, emergent business needs in an organization's business process are local to only that organization. To translate these emergent needs into adaptations of an interaction protocol there is a need to determine how local business needs affect the global view of an interaction. Conversely, changes in business regulations are reflected in adaptations to an interaction protocol which then need to be translated into adaptations of the business processes of interacting organizations. This again calls for relating the global view of an interaction to the local views of interacting organizations. This change propagation process is not unconditional; each participant has to ensure that adaptations to the interaction protocol do not conflict with their business needs.

Thirdly, assuming that adequate solutions are found for the two aforementioned challenges, we are left with yet another challenge which is determining "how" to go about adapting an interaction protocol. The richness of interaction possibilities and the complexity of interaction protocols may yield a large space for adaptation alternatives. Without adequate guidance, the adaptation process may go through many ad-hoc trial and error iterations thereby costing valuable time and money. This calls for a structured adaptation process that facilitates exploring the solution space and guides the stakeholders in making decisions about adaptation alternatives in a systematic manner.

Tackling these challenges is of great importance. Software of the future must be able to cope with a dynamic world where business needs and business context constantly change. Organizations must have the agility that allows them to capture market opportunities and deal with changes in business regulations (Earl F. Ecklund et al., 1996). Two thirds of the cost of software development is spent in modifications that happen after initial development (Swanson & Lientz, 1980; Vilet, 2001) so dealing with these challenges successfully should contribute to non-trivial cost reduction. Furthermore, dealing with these challenges at the level of requirements specification is much more cost-effective (Boehm, 1981).

In this thesis we tackle these three challenges. We propose an approach whereby we guide the collaboration between participants in an interaction, under the supervision of a regulatory agency, on adapting an interaction specification. We propose representing a choreographed interaction at the level of organizational requirements motivating the interaction. We construct a requirements-driven adaptation process that allows reasoning about an interaction from the point of view of each participant to ensure that adaptations satisfy requirements of participants. We propose a technique through which an adapted interaction protocol is obtained from organizational requirements systematically, thereby ensuring correct realization of the requirements.

1.1 Research Questions

From the above motivation we arrive at the following research question:

> *How do stakeholders perform systematic requirements-driven adaptation of a choreographed interaction specification to accommodate emergent business needs and regulations?*

We elaborate the research question into the following list of concrete questions:

1. What representations of a cross-organizational interaction are amenable to systematic adaptation? A main consideration here is to find a representation that allows reasoning about satisfaction of business requirements motivating an interaction. Such a

representation has to support reasoning from the viewpoint of each interaction participant. A secondary consideration is the formality of the requirements representation. The availability of a formal grounding provides precise semantics for relating interaction specification viewpoints. On the other hand, a representation that relies solely on a formal language will be hard to use for practitioners who lack the necessary background (Dwyer et al., 1999).

2. How do participants in an interaction reconcile their business needs when adapting an interaction protocol? The challenge here is twofold. First, business needs of interacting participants are often competing, and the specification of an interaction has to satisfy needs of all participants. Second, a choreographed interaction protocol is constructed from yet another point of view, the global point of view, and hence there is a need to relate the point of view of each participant with the global point of view.

3. What guidance is required for systematic adaptation of an interaction protocol? A systematic adaptation process has to provide guidance on three axes. The first requires providing a catalogue of adaptation operations that, when applied to an interaction specification, produce a valid adapted interaction specification. Second, a wide range of alternatives for adapting an interaction protocol may exist. An adaptation process has to provide guidance to the interaction participants on navigating the space of adaptation alternatives and evaluating them, each from their local point of view. Third, an adaptation process must help in propagating changes back and forth between the global and local views.

4. How can an adapted interaction protocol be obtained systematically from adapted business requirements? On the one hand, requirements models support business-level (or problem-level) reasoning about an interaction. On the other hand, an interaction protocol is an operational representation consumable by machines enacting the protocol. Means for maintaining consistency between these two representations are called for. The challenge is thus in exploiting constraints embedded in requirements models to derive a messaging protocol that realizes the requirements.

1.2 Overview of the Approach

Figure 1.1 depicts an overview of our approach for adapting choreographed interaction specifications. We advocate the representation of a choreographed interaction using requirements models that capture intentions of participants, as described in chapter 4. Requirements models capture local business goals of each participant as well as cross-organizational dependencies motivating them to interact and global contextual constraints on the interaction. The relations that combine the global and local constituents of an interaction into a consistent specification are detailed in chapter 5. In chapter 6, we formalize constraints embedded in requirements models and utilize them to construct a technique for deriving a messaging protocol from requirements models. As new local business needs and global constraints arise, interaction participants collaborate on adapting the requirements models to incorporate these needs, equipped with techniques we provide in chapter 7.

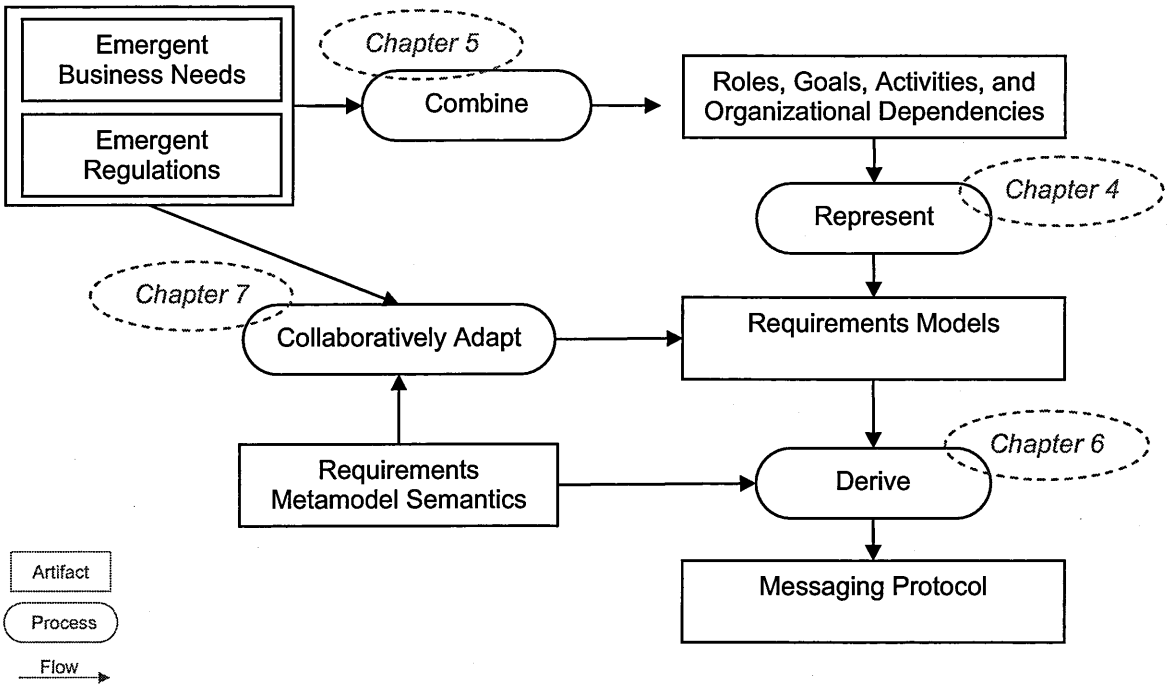


Figure 1.1 Overview of our approach for adapting cross-organizational interaction protocols

1.3 Summary of Contributions

The approach proposed in this thesis presents three novel contributions (Figure 1.2):

1. A formalization of the relation between the global view of an interaction specification and the local view of each interaction participant that allows for determining the impact of change in one view on another.

2. A framework for guiding interaction participants in the exploration of the space of alternatives for adapting an interaction specification, application of adaptation operations, and systematic change propagation.

3. An automated technique for deriving a choreographed messaging protocol from a business-level specification of interaction.

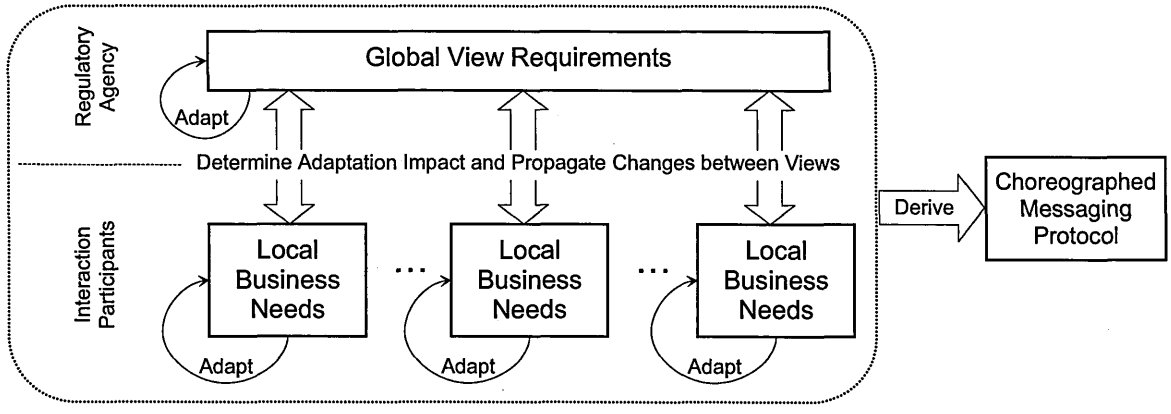


Figure 1.2 Overview of our contributions to requirements-driven adaptation of an interaction

1.4 Research Methodology

Our research was motivated by a general interest in supporting flexible service-oriented interactions. We adopted a methodology commonly used in engineering research (Potter, 2006), which has also been applied in SOC (Kohlborn et al., 2009), and conducted our research by carrying out the following activities:

- Review literature related to service-oriented interactions, especially that which tackles adaptation of interaction protocols.
- Identify open questions that are not fully addressed by existing literature.
- Select a question of interest and identify involved challenges.
- Review literature that attempts to handle these challenges and identify gaps.
- Engineer a solution that builds on success of previous attempts and fills the gaps.
- Validate the solution by applying to case studies and gauge its utility using qualitative criteria extracted from the adaptation literature.

Our research naturally went through iterations, where in each iteration we narrowed down the research questions and refined our solution approach to widen its scope of utility (Cryer, 2000). Feedback that we received on our peer-reviewed publications provided useful literature pointers that helped guide the research direction, which in turn helped with the refinement of our approach. Validation using a case study is common in specification (and adaptation) research (Feather et al., 1997). For validation of our approach, we considered the following case study techniques:

- Construct an exemplar case study.
- Pick an existing case study from the literature.
- Establish a real-world case study.

Constructing an exemplar case study involves coming up with a small enough example to demonstrate the usefulness of the approach. The example is significantly simplified but still exhibits some interesting aspects that are not immediately obvious by inspection (Wing, 1988). We use an example from the healthcare domain to demonstrate our approach throughout this thesis. The simplicity of the example facilitates going through it without a lot of explanation, which would distract from demonstrating our approach. Without enough care, a constructed

example can be deemed contrived or trivial. To mitigate this risk we shared our example in four peer reviewed publications in three different research communities. The main drawback of a constructed example is that it can be tailored with a bias towards demonstrating strengths of an approach and hiding its weaknesses.

Picking an existing example from the literature provides some mitigation of the bias risk. Using a common "yardstick" exemplar to test a family of approaches is a widely used technique (Lewerentz & Lindner, 1995). We chose to tackle an exemplar that is frequently recurring in the interaction specification and adaptation literature. The example, which is drawn from the vehicle insurance domain, provided a more complex case for testing our approach than our constructed example and provided further evidence of its utility.

The ultimate test of an approach is to apply it to a real-world scenario. A use case drawn from the real-world exhibits more detail and randomness to stress-test an approach than examples appearing only in the literature. A real-world example also serves to verify the "External Validity", i.e. that the problem we are tackling is indeed a real one (Yin, 1994). To benefit from the work we have done on the vehicle insurance example, we chose to pick the real-world study from the same domain. We analyzed government regulations for vehicle insurance in several areas in North America, extracted requirements stated in these documents, and applied our approach to them. This helped further prove the utility of the approach, and more importantly, identify aspects that needed improvement.

To support repeatability of our experiments, we implemented a tool for deriving interaction protocols from requirements models, applied it to our constructed example as well as the real world example, and made it publicly available.

It may be argued that analysis of results in our case-study-based evaluation may still be skewed towards our own evaluation criteria. To mitigate this risk, we "benchmarked" our approach using popular evaluation criteria suitable for the field. We evaluated our approach against sets of criteria drawn from the interaction specification and adaptation literature, which helped characterize our approach and position it within the literature.

1.5 Organization of this Dissertation

Chapter 2 covers the necessary background on SOC and motivates our work towards requirements-driven adaptation of service-oriented interactions. Chapter 3 surveys research related to adapting service-oriented interaction protocols and identifies gaps. Chapter 4 describes the representation we adopt for cross-organizational interaction using models of organizational requirements. Chapter 5 modularizes the specification of a choreographed interaction into four viewpoints and puts forward a framework for relating them. Both chapters 6 and 7 build on our framework for relating interaction specification viewpoints. Chapter 6 details our technique for deriving messaging specification from the organizational requirements motivating an interaction. Chapter 7 presents our framework for guiding collaborative adaptation of an interaction specification. Chapter 8 validates our approach using case studies and evaluates it using criteria from the adaptation literature. Chapter 9 concludes and discusses future work. In reference to section 1.1, research question 1 is addressed in chapter 4, question 2 is addressed in chapter 5, question 3 is addressed in chapter 7, while question 4 is addressed in chapter 6.

Chapter 2. Background and Motivation

This chapter covers general concepts of SOC with a focus on service interactions. First, we discuss the emergence of SOC and the vision it provides for building flexible software-intensive systems. We zoom in on service-oriented cross-organizational interactions and state-of-the-art in interaction specification. Next, we discuss requirements specification for choreographed interactions and argue for the need for adapting interaction specification to accommodate emergent requirements. We motivate our work by identifying the challenges of adapting interaction specifications. We explain concepts we introduce with an example that we use throughout the thesis

2.1 Service-Oriented Computing

Today's organizations face rapidly changing market landscape and increasing global competitive pressure (Josuttis, 2007). The expansion of the Internet opened unprecedented business opportunities for businesses to collaborate electronically in a federated manner to provide business value. Connecting to customers, suppliers, and partners electronically became the top IT management issue (CSC, 2000). To succeed in this environment, an organization needs flexible IT infrastructure that can evolve quickly to meet new business demands (Papazoglou et al., 2005). Traditional IT infrastructures where applications are managed and owned by a single enterprise are being replaced by networks of applications whose ownership is distributed across many enterprises (Curbera et al., 2003). In this climate, integration and interoperability became key elements of a flexible IT infrastructure (Papazoglou et al., 2005). The diversity of enterprise IT infrastructure and enterprise applications pose significant interoperability challenges that traditional software development approaches were not designed to face. SOC emerged to take on challenges of autonomy and heterogeneity. SOC promises loose-coupling, implementation neutrality, and flexible configurability by employing standards-based services as the basic building block for enterprise software (Huhns & Singh, 2005).

2.1.1 Standards-Based Services

Services are platform-independent abstractions around heterogeneous components and infrastructure (Huhns & Singh, 2005). Services provide machine-processable interfaces that allow accessing of business functionality over a network by exchanging messages (Booth et al., 2004). At the heart of SOC is information hiding (Parnas, 1972) where the heterogeneity of enterprise platforms and applications are hidden behind message-oriented service interfaces. Service interface are specified using standards-based languages thereby providing implementation-neutrality.

The expansion of the Internet and wide adoption of Web and XML (Bray et al., 2004) technologies provided an adequate foundation for fulfilling the SOC promise through "Web services". A Web service is a realization of a service whose description and transport utilize open Internet standards (Papazoglou & Georgakopoulos, 2003).

Web service technologies enable interoperability via standards-based specification of service interfaces, message formats, and communication protocols (Cerami, 2002; Curbera et al., 2003). The Web Service Description Language (WSDL) emerged as a standard for describing operations provided by a service interface (Christensen et al., 2001). Communicating with service operations is achieved via XML messages, typically sent and received over the widely standardized HTTP protocol. XML messages contents are encoded using standardized formats such as the Simple Object Access Protocol, or SOAP, (W3C-SOAP, 2007). Ultimately, message and service semantics shall also be described in standard formats to enable semantic-level interoperability (Martin et al., 2004).

2.1.2 Service-Oriented Architectures

Service-Oriented Architecture (SOA) is a set of principles that guide the design of service-oriented systems. SOA promotes loose coupling, coarse-grained service interfaces, and self-containment of services (Kaye, 2003; Erl, 2004). Coarse-grained service interfaces encapsulate IT infrastructure and expose business-level operations that have a business-relevant meaning (Acharya et al., 2005). SOA promises to exploit this alignment between services and business at

runtime by enabling dynamic service binding. This dynamism is facilitated by interposing a registry as a discovery agent (Kaye, 2003). Intermediating a registry, a "Service Bus" (Josuttis, 2007), or other intermediaries facilitates loose coupling between service providers and consumers. Self-contained service descriptions are published to a service registry. Clients discover services that match their needs by looking them up in a registry. Clients bind dynamically to services at invocation time, thereby providing flexibility in choosing a service provider, possibly on a per-invocation basis (K.H.Bennett et al., 2001).

2.1.3 Service Composition and Coordination

Composing "large pieces" of software out of existing pieces is a recurring challenge in software engineering (Garlan et al., 1995). This challenge is compounded when the composed pieces exhibit interfaces that are specified in different languages with disparate syntax and semantics (DeLine, 1999). By employing standards-based service interfaces, e.g. using WSDL, SOC avoids the interface mismatch problems and supports integrating services into complex systems (Burdett & Kavantzas, 2004). The literature has typically distinguished between two flavors of service composition specification (Peltz, 2003):

- Orchestration: Service composition as a centrally controlled executable process flow.
- Choreography: Decentralized, peer-based specification of the observable interaction protocol between a set of collaborating services.

2.1.3.1 Orchestration

Orchestration refers to how services are composed into an executable process that specifies data and control flow between them from one entity's point of view, i.e. an orchestrator. Orchestration is suitable for specifying an organizational "business process", i.e. a specification that weaves business activities of an organization or an organizational unit into a workflow. A business process can itself be exposed as a composite service thereby enabling complex compositions (Khalaf et al., 2003). A slew of composition standards emerged over the past decade (van der Aalst, 2003), culminating in the adoption of the Business Process Execution

Language for Web Services (BPEL4WS), or BPEL for short, (Andrews et al., 2003) as the de facto standard for service orchestration both in literature and industry.

2.1.3.2 Choreography

Choreography is concerned with specifying the external observable behavior between multiple participants, i.e. services and clients. External observable behavior is defined as the presence or absence of messages exchanged between participants (Austin et al., 2004). Choreography specifies what messages can be exchanged, conditions under which participants can exchange messages, and allowed exchange sequences (Booth et al., 2004). As such, choreography is concerned with specifying an "interaction protocol" between participants. As opposed to orchestration where services are composed from the local point view of one entity, choreography is concerned with messaging specification from a "global" point of view, i.e. a neutral-observer point of view. Choreography is suitable for describing multi-participant interactions that are overseen by a neutral entity, such as a regulatory agency. In what follows we zoom in on choreographed interactions and their specification.

2.2 Choreographed Service-Oriented Interactions

An interaction between multiple participants is said to be "choreographed" if the protocol governing the interaction is described from a neutral point of view. A choreographed interaction protocol specifies allowed messages and message sequences between a set of "roles" representing abstract participants. To support interoperability, interaction protocols are specified in an implementation-independent standard form. A protocol specification is a publishable document to be made available for participants wishing to play a role in an interaction. Actual participants enact the protocol by implementing services and clients that abide by its specification.

Choreography plays a central role in coordinating business within a "business community", i.e., a collection of organizations that share a common market sector. Interactions between organizations in a community are regulated by a neutral entity. For example, a port authority

may regulate the interaction between shipping agencies, terminal operators, customs brokers, transport operators, and other business operators within a port (Baglietto et al., 2002).

2.2.1 Specifying Choreographed Interactions

Two styles of choreography description languages emerged: "Interaction style" and "Interconnection style". In interaction style, elementary messaging interactions, i.e. sending/receiving a message, are the basic building blocks. An interaction protocol specifies behavioral dependencies between these interactions, and compositions thereof, using control flow constructs. In the realm of Web service standards, the leading interaction-style choreography specification language is the Web Service Choreography Description Language (WS-CDL) which we willl present in section 2.2.3.

In interconnection style, control flow is defined per participant, i.e. from a local point of view, and the local flows are combined together using message links (Decker et al., 2008). The graphical notation of Business Process Modeling Notation (BPMN) (White, 2004) is gaining popularity in interconnection-style choreography specification (Decker & Barros, 2008). Local behavior of each participant is represented as a swimlane encapsulating their business activities and sequencing between them. Connections between participants are captured via "message links" that connect activities in different swimlanes. BPMN is useful for capturing inter-connection between related business processes, but it does not provide a specification of a global messaging protocol. In fact, local participant behaviors may be incompatible and a messaging protocol may be unattainable (Decker & Barros, 2008). For these reasons, we adopt interaction-style representation of choreography, which we now present an example of.

2.2.2 Running Example – Medical Choreographed Interaction

Throughout the rest of this thesis we use a choreographed interaction from the medical domain as a running example. The interaction involves three roles: Patient, Medical Provider (MP), and Doctor (Figure 2.1). Messaging between these roles is choreographed as follows. A Patient who needs to visit a Doctor is required to get an authorization from her MP first. When the Patient has received authorization for treatment from the MP, she can request an appointment

from a Doctor. After getting an appointment confirmation the Patient visits the Doctor to get examined. The Doctor later sends a prescription to the Patient, bills the MP, and gets back an electronic payment. The neutral observer in this interaction is the state's health department that oversees interaction between participants and handles disputes.

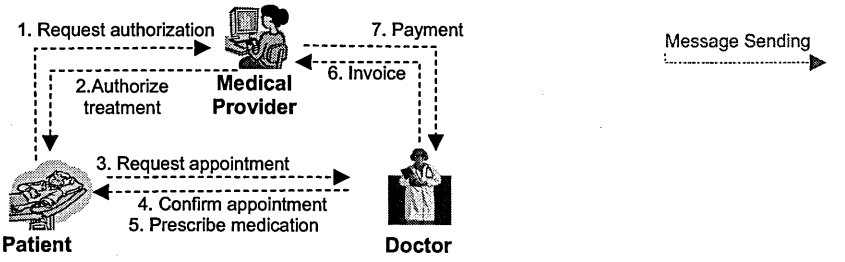


Figure 2.1 Choreographed messaging for the running medical example

The messaging protocol regulates an abstract interaction between the three roles. At runtime, any number of participants can play a role in any number of instances of the interaction. For example, many patients are expected to request authorization from an MP and request appointments from many different doctors. The protocol constrains messaging that may take place between participants at runtime. For example, the protocol specifies that a patient may not request an appointment unless she has obtained an authorization from her MP. In a choreographed interaction, participants interact in such a way that none of them observes all messages being exchanged, yet interactions taking place between some participants affect the way other participants interact (Zaha et al., 2006b), which justifies the need for a global view. A Doctor's decision to accept an appointment request depends indirectly on the exchange between the Patient and the MP for authorizing treatment.

2.2.3 Web Service Choreography Description Language (WS-CDL)

In the sphere of Web services standard languages, WS-CDL is the leading standard for specifying choreographed interactions (Barros et al., 2006). WS-CDL is an "XML-based language that describes peer-to-peer collaborations of Web services participants by defining, from a global viewpoint, their common and complementary observable behavior, where ordered message exchanges result in accomplishing a common business goal" (Kavantzas et al., 2005).

WS-CDL specifies a choreographed interaction as a composition of activities. The elementary activity is message send/receive. Composite activities are specified using "Sequence" and "Parallel" control-flow constructs. A "Choice" activity provides conditional choice between mutually exclusive paths. Conditional repetition is supported through a "Workunit" activity (Barros et al., 2005a). An abridged version of the WS-CDL specification of the medical interaction is shown in Figure 2.2.

```
<sequence>
  <interaction name="AuthorizationRequest">      <participate fromRole="Patient" toRole="MP"/>     </interaction>
  <interaction name="TreatmentAuthorization">    <participate fromRole="MP" toRole="Patient"/>     </interaction>
  <interaction name="AppointmentRequest">        <participate fromRole="Patient" toRole="Doctor"/> </interaction>
  <interaction name="AppointmentConfirmation"> <participate fromRole="Doctor" toRole="Patient"/> </interaction>

  <parallel>
    <interaction name="Presription"> <participate fromRole="Doctor" toRole="Patient"/> </interaction>
    <sequence>
      <interaction name="Invoice">  <participate fromRole="Doctor" toRole="MP"/> </interaction>
      <interaction name="Payment"> <participate fromRole="MP" toRole="Doctor"/> </interaction>
    </sequence>
  </parallel>

</sequence>
```

Figure 2.2 Partial WS-CDL specification of the medical interaction protocol

2.2.4 Formal Groundings of Choreography

The primitive constructs of choreography languages have been guided by previous work on coordination languages (Arbab, 1996) and the semantics of distributed process specification languages. Process calculi, such as CSP (Hoare, 1985), describe in a formal way the collective behavior of a set of distributed communicating processes. Each process exhibits a number of legal states. In each state, a process is allowed to communicate certain data with other processes over a communication channel. Communication between two processes occurs via "shared events", i.e. events observable to both processes, such as exchanging a message. Once the communication occurs, a process may transition to another state. For example, after sending an authorization request the process corresponding to the Patient role enters a "waiting for authorization" state in which it is not allowed to send messages. Once a treatment authorization message is received the process enters another state in which it is allowed to send an appointment request. At any point in time, the state of a choreographed interaction is defined as the combined state of all participant processes.

Since a choreographed interaction may be modeled as a set of communicating distributed processes, choreography description languages have been influenced by the literature on process calculi. One process calculus in particular, the PI calculus (Milner, 1999), influenced the design of WS-CDL (Carbone et al., 2007). A distinguishing aspect of the PI calculus is treating the channel itself as data, allowing a process to send a channel to another process. In effect, this allows treating services (or participants) as data. For example, if desired, an MP can specify the Doctor which a Patient has to communicate with. However, a global view of an interaction is less pronounced in PI calculus than in WS-CDL (Decker et al., 2006a).

Other formalisms used for representing interactions include Petri Nets (Petri, 1962), albeit they do not provide an explicit global view (Su et al., 2008), and event calculus (Kowalski & Sergot, 1986) which is missing the representation of roles (Khaled & Spanoudakis, 2004).

2.3 Specification of Requirements for Choreographed Interactions

The success of a software system depends on how well it fits needs of its stakeholders and its environment (Nuseibeh & Easterbrook, 2000), and choreographed interactions are no exception. Software requirements comprise the needs of stakeholders, and Requirements Engineering (RE) is the process by which the requirements are determined. Successful RE involves "understanding the needs of users, customers, owners, and other stakeholders; understanding the contexts in which the to-be-developed software will be used; modeling, analyzing, negotiating, and documenting the stakeholders' requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution" (Cheng & Atlee, 2007).

Most importantly, RE is concerned with the real-world goals for, functions of, and constraints on software systems as well as the relationship of these factors to precise specifications of software behavior, and to their evolution over time (Zave, 1997). Thus, understanding and analyzing real-world goals of stakeholders in a choreographed interaction is crucial to successful adaptation of the interaction specification.

2.3.1 Goals of Stakeholders in an Interaction

A "goal" is an objective statement of what a software system under consideration should achieve (van Lamsweerde, 2001). Goal formulations are "optative" statements that refer to intended properties to be ensured; as opposed to indicative properties of the world (Zave & Jackson, 1997). For example, in the medical interaction, "get healed from ailment" is the goal of a Patient and "generate profit from medical practice" is a goal of a Doctor. Goals are formulated at different levels of abstraction ranging from high-level, strategic concerns to low-level technical concerns. Elaboration of high-level goals into low-level goals and eventually architectural and operational specification is referred to as "Goal Refinement". Goal refinement provides traceability from high-level strategic objectives to solution specification, thereby providing linkage between organizational and business context to the software being constructed (Yu, 1993). For example, the Doctor's goal of profiting from practice is refined into sub-goals pertaining to conducting treatment as well as managing finances. These goals are eventually operationalized, in the particular scenario specified by the medical interaction, into activities for scheduling appointments, examining patients, and billing. Refinement techniques provide stakeholders with means to explore and evaluate choices between alternative means for achieving their goals (van Lamsweerde, 2000). For example, during refinement, a Doctor may consider to carry out billing activities electronically or via paper documents.

Goals have long been recognized as essential components in the RE process (Ross & Schoman, 1977) and particularly for tackling the "why", "what", and "who" aspects of a software system (van Lamsweerde, 2000):

- Why aspects: Goals capture the motivation behind an envisioned system. As such, goals models provide rationale for requirements and facilitate explaining requirements to stakeholders (Sommerville & Sawyer, 1997).

- What aspects: Goal models provide means for sharing knowledge among stakeholders and help ensure that requirements specification for a system to-be-developed are

complete (Yu 1987). Through refinement, goals are operationalized into specifications of services to be implemented and constraints to be enforced.

- Who aspects: Goal models support reasoning about the assignment of responsibilities for the resulting requirements to agents including humans, devices, and software.

In a service-oriented interaction, participants delegate the achievement of their goals to software agents, i.e. services and service clients, that carry out the interaction on their behalf. To ensure that stakeholders goals are met, it is key to understand and precisely specify the behavior of these agents and the collaboration between them.

2.3.2 Requirements-Driven Multi-Agent Interactions

An agent is a "computer system situated in some environment, and that is capable of autonomous action in order to meet its delegated goals" (Wooldridge & Jennings, 1995). Multi-Agent Systems are systems built out of distributed, autonomous, coordinated agents. Agent autonomy provide the flexibility required in open systems in which constituent components and their interaction patterns constantly change (Zambonelli et al., 2003). Services and service clients are typically implemented by agents (Booth et al., 2004) and cross-organizational interactions can be viewed as a multi-agent system (Huhns et al., 2005). Agent-oriented methodologies provide notations and techniques for specifying and reasoning about agent inter-dependencies, agent interactions, and requirements that motivate these interactions. By representing a choreographed interaction as the coordination between a set of agents, we can reason about achievement of stakeholder goals and their relation to agent interactions.

Agent interactions are specified in terms of externally observable behavior of agents. In addition to external behavior, each agent embodies internal business policies that govern its operation. Agent specification must take into account consistency between external and internal behavior. Additionally, agent specification has to address how agent behavior achieves its delegated goals. A specification of all these concerns can become quite complex. Separating these concerns using specification "viewpoints" serves to modularize and manage complexity of an interaction specification.

2.3.3 Modularizing Interaction Specification Using Viewpoints

A "concern" is any interest in a system relevant to one or more of its stakeholders (ISO/IEC 42010). Separation of concerns was established in early software engineering history as a means for managing complexity (Dijkstra, 1974). The principle of separation of concerns lead to the use of multiple viewpoints for describing and reasoning about software systems (Lago et al., 2010). A viewpoint typically embodies one aspect of a software system requiring its own notation and reasoning techniques. The notion was popularized (O'Leary, 2009) by a proposal for using viewpoints to model software architecture (Kruchten, 1995).

Choreographed interactions are poised to benefit from a multi-viewpoint specification for managing complexity and modularizing an interaction specification. Whereas operational aspects of an interaction are specified in terms of messaging, a specification of stakeholder goals is also needed to support business-level reasoning (section 2.3.1). The need for two different representations implies that the specification of these two aspects may be separated into different viewpoints. Furthermore, viewpoints can help manage the complexity of interaction specification by separating the specification of external and internal behavior of a participant.

Having separated concerns into viewpoints, it is typically necessary to integrate them into a unified whole (Jackson, 1990). The behavior of a software agent constructed to act on behalf of an interaction participant must comply with the expectations of other participants as well as with that participant's internal business policies. For example, an agent that implements appointment scheduling at a Doctor's office must provide appointments to Patients, in accordance with the specification of the medical interaction, and at the same time take into account the schedule of staff and hours of operation at the Doctor's office. Viewpoints make possible the construction of separate, yet consistent, specifications of these aspects. Techniques for combining viewpoints aim to ensure consistency between them especially when they can evolve separately under the ownership of multiple stakeholders (Nuseibeh et al., 1994).

2.4 Adaptation of Choreographed Interactions

Maintenance is an inevitable part of the lifecycle of any software system. Two types of changes amount to 75% of the total cost of maintenance; namely "perfective" and "adaptive" (Swanson & Lientz, 1980). Perfective changes are triggered by new or modified stakeholder requirements while adaptive changes refer to changes in the environment of the software system (Swanson, 1976). For a choreographed interaction the two types are manifested in changes in business needs driving participants to interact and changes in regulations governing the interaction, respectively. Consequently, an interaction protocol has to be adapted to reflect these changes.

Consider an emergent need to protect an MP from abuse of coverage, i.e. the need to ensure that an MP covers treatment expenses only for eligible patients. To achieve this business goal an MP requires a Doctor to collect a Patient's Medical Plan ID (MPID) so that the MP may check its validity. The MP will not hold itself liable for covering treatment expenses unless a Doctor verifies Patient MPID before submitting an invoice. This requirement imposes a constraint on the order in which a Doctor performs his activities. One might expect that a realization of this requirement involves having a Doctor send a "Verify coverage" message to an MP at some point in time before sending an invoice. It is hard to rationalize an adaptation to a messaging protocol without relating the protocol to business goals it is meant to achieve. Additionally, since multiple participants are involved in the interaction, it is necessary to determine the impact of any adaptation of the protocol on their goals. If a participant finds a suggested adaptation unacceptable from their point of view, they should be able to collaborate with the rest of participants on finding acceptable alternatives. These challenges facing adaptation of an interaction protocol are summarized into the following three categories.

2.4.1 First Challenge - Bridging Business Needs to Interaction Specification

Current standards for representing choreography, e.g. WS-CDL, specify operational aspects of a choreographed interaction in terms of message exchanges along with control and data flow. As such, these representations are detached from business-relevant aspects, mainly goals

motivating an interaction between participants. Without connecting messaging exchanges to business goals motivating the exchanges, it is hard to argue that an adaptation of an interaction protocol satisfies these goals.

The problem is exacerbated by another deficiency in emerging standards. A messaging protocol is only part of a wider view of an interaction. Activities performed by humans outside of electronic messaging often are crucial to achieving business goals of an interaction. For example, the medical interaction example is meaningless if a Doctor does not physically examine a Patient, which can only be made possible if a Patient physically visits a Doctor's office. WS-CDL does not facilitate including either of these activities in the specification of Figure 2.2.

Whereas WS-CDL is useful as a machine-readable standards-based language for specifying messaging protocols, there is a need for a representation that explicitly captures business goals motivating an interaction. Such a business-level representation would facilitate reasoning about business goals and their fulfillment, in terms of a messaging protocol combined with physical activities. The need for two different, yet complementary, representations of an interaction calls for means to maintain consistency between them (Krishna et al., 2009).

Thus, the first challenge lies in finding a representation of a choreographed interaction suitable for business-level reasoning and relating it to the specification of a messaging protocol in a way that allows us to maintain consistency between the two representations.

2.4.2 Second Challenge - Reconciling the Needs of Multiple Stakeholders

The driver for evolving a choreographed interaction can be either local or global. Local business needs of any participant may change thereby leading to a change in their business process. Local changes may have consequences on the global messaging protocol, and therefore there is a need to identify these consequences and propagate the changes (Bohner & Arnold, 1996).

On the other hand, a regulatory agency overseeing an interaction may decide to change rules that govern an interaction. Changes in rules are reflected as changes in the messaging protocol,

which may then have an impact on each of the participants' local views. Therefore there is also a need to propagate changes from the global to local views.

Whether the driver of change is local business needs or the global interaction context, effects of the change need to be propagated to all affected views to maintain consistency between them. That is, changes from one participant's local model need to be translated into changes in the global model which may then translate into changes in another participant's local model, and so on.

A further complication arises from the shared-ownership nature of a choreographed interaction. A choreographed interaction is an anarchic system (Chandy & Rifkin, 1997) where responsibility of design and implementation is shared among multiple entities, namely representatives of the interacting roles as well as the global observer (e.g. regulatory agency). Any change propagation mechanism has to enable the participants and the regulatory agency to collaborate on the design of the adapted interaction protocol.

The second challenge thus lies in establishing relations between the global view and local views to enable bidirectional change propagation and maintain consistency between them as well as allow participants to collaborate on reaching an agreement on a messaging protocol.

2.4.3 Third Challenge - Providing Guidance for Disciplined Adaptation

Associating engineering methodologies with process/interaction modeling techniques is one of the grand challenges facing SOC (Papazoglou et al., 2006). Assuming the two aforementioned challenges are adequately addressed, we are left with the challenge of how to go about performing an adaptation. Ad hoc adaptation is likely to result in erroneous results thereby wasting time and money. Systematic and disciplined means for adapting all interaction specification artifacts are thus necessary (Mens & D'Hondt, 2000).

Given an emergent business requirement and a specification of an interaction, we need to guide architects on how to incorporate the new requirement into the specification in a systematic manner. At the very least, we need to ensure that an adapted interaction specification is

structurally and semantically valid, free from internal inconsistencies, and free from anomalies such as deadlocks (Decker & Barros, 2008).

For non-trivial interactions, the space of alternatives for incorporating an emergent requirement can be quite large. Automated means for assisting architects with exploring and evaluating alternatives are thus called for. Additionally, to maintain consistency between views we need systematic means to guide change propagation. This calls for means for assessing the impact of a given change in one view on other views. Propagating changes to affected views then guides progression of adaptation in a disciplined manner.

The third challenge thus lies in providing guidance for adapting an interaction specification and propagating changes to all affected views to maintain the validity of an interaction specification and its internal consistency.

2.4.4 Other Challenges

There are several challenges other than the three we have now detailed. Some of these challenges are significant and warrant substantial research beyond the scope of our work. To further scope the problem we are tackling, we briefly discuss the challenges that we will not address in this thesis.

2.4.4.1 Enabling Runtime Adaptability

In this thesis, we focus on design-time adaptation of choreographed interactions. An ambitious goal of software engineering and systems research is to enable "self-management". A self-managed system is one that can autonomously control its own configuration, resource allocation, or security settings (Herrmann, 2005). A self-managed system is envisioned to self-adapt in response to changes in its environment (Martin-Flatin et al., 2006). This topic has been tackled from several angles. It is argued that the main purpose of agent-oriented methodologies is enabling this dynamic flexibility (Winikoff, 2009), where an agent adjusts its plans in response to changes in its environment. Also, a major part of the promise of SOA/SOC is to enable runtime configurability (Nitto et al., 2008). This research area is still emerging and many significant challenges remain. Change management in a distributed environment (Kramer &

Magee, 2007) is particularly relevant to change propagation in a choreographed interaction. Agreement on a protocol for a choreographed interaction involves negotiation between stakeholders representing each role. Enabling services to negotiate autonomically on behalf of stakeholders is a major challenge, especially considering the need to maintain a consistent view between many participants (per role) participating in the protocol.

2.4.4.2 Dealing with Running Interaction Instances

When a new (i.e. adapted) protocol is enacted it replaces a protocol that may already be in use. For a long-running interaction, instances of the interaction that follow the existing protocol may already be in progress at the time when the new protocol is enacted. There is a question of what to do with these instances

- Assume that no instances are running when a new protocol replaces an existing one.
- Terminate and abort all running instances
- Let any running instances run to completion under the old protocol
- Dynamically migrate running instances to the new protocol

The latter alternative is the most complex as it requires making changes to the structure and the state of a running instance correctly and efficiently (Rinderle et al., 2004). Although interesting research questions arise, such as ensuring compliance of an adapted instance (Reichert et al., 2009), these set of questions are orthogonal to the three challenges tackled in this thesis.

2.4.4.3 Supporting Service-Level Agreement

A Service-Level Agreement (SLA) is a contract that associates financial and legal obligations of interacting parties with observable qualities of service (Lamanna et al., 2003). Quality of service (Qos) includes so called "non-functional requirements" such as reliability, availability, and performance. QoS is generally specified as constraints on the characteristics of service operation execution. Conventionally, QoS obligations are described separately from obligations specified by a messaging protocol. Several languages for specifying QoS characteristics have been proposed (Emmerich et al., 2003) emerging both from the industry, such as WSLA (Ludwig et al., 2003), and from research, such as HQML (Gu et al., 2001). Attempts to provide

semantics for QoS reinforce this separation. Notably, Skene et al. (2004) specify QoS characteristics in terms of performance (e.g. average latency), reliability, backup frequency, and granularity of monitoring data. QoS characteristics may evolve separately from a messaging protocol; SLAs can be established and terminated while the services in question continue to exist independently (Skene, 2007). Although, non-functional requirements have been used to rationalize design decisions and drive the adaptation of a single-owner system (Chung et al., 1995), their relation to functional aspects is not well-understood in the context of multi-participant interactions (Desai et al., 2009).

2.5 Chapter Summary

This chapter covered background on SOC with a focus on choreographed interactions. We presented a brief overview of SOC concepts, its origins, and its promise for interoperability. We contrasted service orchestration, as a centrally controlled service workflow, and choreography, as an enabler for inter-organizational interoperability. We zoomed in on choreographed interactions and introduced state-of-the-art languages for choreography specification using our running medical example. We discussed requirements for a choreographed interaction viewed as a set of distributed agents interacting to achieve business goals. We concluded by scoping the problem of adapting choreographed interactions and the challenges that have to be faced. The next chapter surveys existing work that has attempted to tackle these challenges.

Chapter 3. Related Work

We identified, in section 2.4, three categories of challenges facing adaptation of choreographed interactions: namely bridging requirements specification to messaging specification, relating and reconciling requirements of interaction stakeholders, and providing guidance for a disciplined adaptation process. In this chapter we seek solutions for these challenges in the literature from various areas of software engineering. This chapter summarizes research that attempted to tackle the challenges we identified. We examine this work through three lenses, one corresponding to each category of challenges. At the end of the chapter we summarize gaps that we identified in existing literature with respect to each challenge and indicate how we will address these gaps.

3.1 Bridging Representations across Levels of Abstraction

A messaging specification is a machine-readable specification of operational behavior in terms of messages exchanged between participants. The need to capture stakeholders concerns and specify the business problem calls for representing business concepts, problem context, and stakeholders' requirements. Requirements models are useful for communicating and reasoning about stakeholders goals at a level suitable for architects and analysts. Requirements specifications exhibit a higher level of abstraction than messaging specifications, they describe what problem is being solved and why, rather than how it is being solved (Jackson, 2001). The need for two disparate representations at different levels of abstraction calls for means to relate them and keep them consistent. This section examines literature that attempts to capture requirements motivating an interaction and relates them to a messaging specification, or more generally relates representations of an interaction at different levels of abstraction. This area of work has been conventionally divided into three sub-categories: moving from higher to lower levels of abstraction, moving from lower to higher levels of abstraction, and integrating high and low level representations together (D. Berry et al., 2003).

3.1.1 From Requirements to Services

Conventional software engineering wisdom called for progressing from requirements to implementation, i.e. from high to low levels of abstraction (Benington, 1987). SOC standards are almost all about operational specification, and deriving such operational specifications from higher-level specifications is a frequently recurring need (OMG-SOAML, 2009). We survey research that attempts to obtain service interfaces and interaction specifications from high-level specifications in systematic ways. We categorize these approaches by the type of high-level representation they employ, namely: business-concept models, business rules, business goals, business interconnections, or a combination of goals and interconnections.

3.1.1.1 Conceptual Modeling and Model-Driven Transformation

By focusing on operational aspects of an interaction, i.e. messaging, SOC standards are missing representation of business concepts. Conceptual modeling languages, and in particular UML (OMG-UML, 2004), provide notations and techniques for capturing business concepts and their relations for a software system or a business domain. Representing service-oriented systems at a conceptual level raises the level of abstraction to a level closer to the business problem from the messaging specification. Furthermore, UML and associated technologies (OMG-MOF, 2010) and (OMG-QVT, 2002) enable automated Metamodel-Driven (MD) transformation between concepts at different levels of abstraction (Mellor et al., 2004). In particular, MD transformation enable automated derivation of operational specifications from conceptual models (Skogan et al., 2004). Service metamodeling can thus be used as a means for establishing relations between business concepts and operational service specifications. Skene et al. (2003) propose a metamodel that encompasses business level concepts as well as service-implementation-level concepts. The metamodel aims to raise the level of abstraction for modeling service-based systems. It distinguishes between "capabilities" of a business service and "offerings" of an implementation-level service, thereby allowing representation of electronic service contribution to business functions. However, the metamodel does not explicate service interactions and no provision for obtaining interaction protocols from business-level models is provided.

Automatic derivation of service interaction specification from conceptual models has been later addressed by Skogan et al. (2004). Service compositions are captured in UML and transformations from the UML representation to a number of currently popular XML-based languages are proposed. UML is used as an implementation-neutral representation to shield designers from having to learn peculiarities of an interaction specification language that might get obsolete quickly. As Van der Aalst (2003) argues, none of the existing XML-based Web interaction specification languages has emerged as a clear winner. Thus, until a de facto standard is established, it is desirable to ease switching from one language to another. However, this approach provides only notational, rather than conceptual, abstraction. The UML representation of an interaction maps directly to messaging protocol concepts, rather raising the level of abstraction by capturing business concepts.

Almeida et al. (2003) achieve a higher level of abstraction via a 3-layer approach to service development. The goal of the approach is to liberate architects from constraints imposed by an implementation platform. A "platform-independent service design" layer is introduced to abstract an underlying service runtime platform, thereby allowing services to be designed for multiple platforms. Although addition of this layer adds design flexibility, it does not help with adaptation of service interactions. By enabling replacement of the runtime platform, in effect, this approach facilitates evolution of the implementation platform rather than service interactions.

These approaches share the benefit of being accessible to practitioners familiar with UML and related technologies. However, their aim is to shield architects from platform and implementation details, rather than capture stakeholders' concerns. Hiding syntax and semantics of service implementation languages in an intermediate conceptual representation layer, though useful, does not solve the problems of capturing problem-level constraints governing an interaction and translating them into an operational interaction specification.

3.1.1.2 Business Rules

Business rules provide means for capturing contextual, structural, and behavioral constraints governing a service-oriented system. Behavior of participants in an interaction can be specified

as a conjunction of constraints on occurrence (or absence) of events and their ordering (Dwyer et al., 1999). Rules capture constraints on receipt of messages, their relative order, and upper/lower bounds on the number of occurrences. Advocates of rule-based representation argue that flow-oriented languages, such as WS-CDL, provide an over-constrained interaction specification by prescribing a fixed sequence of messages. On the other hand, the declarative nature of rules provides flexibility in specifying interactions (Pesic & van der Aalst, 2006). Representative of these approaches is DecSerFlow where graphical constructs are used to represent constraints on receipt and ordering of messages (van der Aalst & Pesic, 2006). A conjunction of DecSerFlow constraints is translated into an automaton whose execution is used to validate execution traces of corresponding service interactions. DecSerFlow does not distinguish between the global and local views of an interaction and does not provide an explicit representation of the local viewpoints of interaction participants. It is thus hard to reason about an interaction specification, or adaptations thereof, from the point of view of a participant.

This deficiency is remedied by Berry & Milosevic (2005) where rules are incorporated into the specification of a choreographed interaction while providing explicit separation between the global and local views. In this approach, Business Contract Language (BCL) is used to express business rules as Permissions, Obligations, and Prohibitions, and Choreography is specified using a language called Finesse. Finesse specifies local behavior of autonomous participants as well as bindings between their visible behaviors. A mapping is provided from BCL constructs to Finesse thereby enabling the incorporation of business rules into choreography. The approach facilitates monitoring execution of an interaction and enforcing business rules, but it does not provide guidance on adapting interaction specification. Furthermore, the Finesse choreography description is assumed to be designed separately from business goals that motivate participants to interact.

The declarative nature of rules allows incremental specification of interaction thereby facilitating adaptation via addition of rules (Li et al., 2005). Even though flow-oriented specifications may over-constrain behavior (Pesic & van der Aalst, 2006), they are useful for interoperability purposes. It is generally hard to derive a standards-based flow-oriented

specification from a set of rules. More importantly, rule-based approaches are not concerned with the origin of the rules, i.e. how business rules arise from stakeholder goals. These approaches are more suited for monitoring interaction execution rather than reasoning about its adaptation.

3.1.1.3 Business Goals

Goals play an important role in software development activities, especially in supporting evolution. Relating goals to activities that realize these goals and services that realize these activities facilitates reasoning about evolution of business needs (Vasconcelos et al., 2001). Goal-oriented methodologies provide techniques for capturing, specifying, reasoning about, refining, and operationalizing goals. These methodologies provide formal patterns for systematically refining goals and eventually deriving architectural and operational specifications which realize the goals (Darimont & van Lamsweerde, 1996).

Representative of these methodologies is Knowledge Acquisition in Automated Specification (KAOS) (van Lamsweerde et al., 1991; Dardenne et al., 1993). KAOS provides a tool-supported means (Bertrand et al., 1997) for deriving functional and architectural specification from goals. Derived architectural specifications are refined to meet domain-specific architectural constraints and non-functional goals (Lamsweerde, 2003). To the best of our knowledge, no attempts have been made to derive service interactions from KAOS goals models, even though it can be argued that deriving event-based transition systems is a step in that direction (De Landtsheer et al., 2004; Letier et al., 2008). More importantly, KAOS adopts the notions of "system-to-be developed" and "system goals" which targets a more centralized development process. On the other hand, choreographed interaction specification requires participation of multiple independent stakeholders with possibly competing sets of goals. Furthermore, although the metamodel underlying the KAOS methodology (Dardenne et al., 1993) represents agents and their goals, it lacks explicit representation of their inter-dependencies on one another for goal fulfillment (Lamsweerde, 2004). Without explicating participant inter-dependencies it is hard to reason about their interaction or derive a messaging protocol that realizes their goals.

3.1.1.4 Business Interconnections

Explicating participant inter-connections is essential for reasoning about adaptation of inter-enterprise interactions. Process calculi such as CSP and PI calculus (section 2.2.4) capture inter-connections between participant processes via shared observable events. These languages are useful for formally specifying and verifying an operational representation of interaction. However, they are detached from the business meanings of inter-connections, and hence not suitable for requirements-driven adaptation without linking to a complementary business-level representation. Approaches that elevate representation of participant inter-connections to the business-level are called for. Business-level inter-connections are captured via shared business activities or business commitments.

Activity-based Interconnection

Representative of activity-based interaction approaches is the graphical Interaction Systems Design Language (ISDL) and associated techniques. ISDL captures participant local behavior as business activities and control flow between them. Participant behaviors are composed into an interaction by interconnecting local models via "interaction activities", e.g. send-receive a message. In this respect, ISDL is similar to BPMN (section 2.2.1), albeit ISDL aims to specify an interaction at multiple levels of abstraction (Dick Quartel et al., 2002). Business models as well as service design and implementation models are mapped to ISDL (Dick Quartel et al., 2005) and consistency between them is verified.

The ISDL-based service design method outlines a sequence of steps for proceeding from business-level specification of an interaction to service design and implementation (Dick Quartel et al., 2004). However, the method does not provide specific guidance on how to perform the refinement of the business-level specification. Some consistency-preserving refinement operations were proposed (Almeida et al., 2005) that allow systematic refinement of activities and behavior interconnections, but only at an operational level. Through refinement, a BPEL specification for local behaviors can be obtained (Dick Quartel et al., 2005), but it is not clear, as in BPMN, how to obtain a choreographed messaging specification from the interconnected participants' models. This task gets harder considering that physical activities are

not captured in an ISDL specification of an interaction. The starting representation of the method is a business process, and origination of the process from business goals is not addressed, let alone goals of multiple stakeholders. Additionally, propagating the impact of changes made in one local view to others is not addressed.

Commitment-based Interconnection

Commitments are directed obligations (Tan & Thoen, 1998) from a debtor role to a creditor role (Singh, 1999). A debtor is obliged to satisfy their commitments towards a creditor if pre-conditions on the commitments hold. For example, the tuple CC(Buyer, Seller, Pay, GoodsDelivered) denotes a commitment from a buyer role to pay for goods if the seller delivers the goods. Commitments are manipulated through operations such as "Create", "Discharge", "Release", "Assign", and "Delegate". A commitment is discharged when a debtor satisfies the commitment or released when the debtor is no longer required to fulfill it. Delegation and assignment shift the role of the debtor or creditor to another role, respectively. Commitments may be nested to represent responsibilities that are pre-conditioned on the satisfaction of other commitments (Desai et al., 2007).

A commitment protocol declaratively captures rules that govern an interaction between a set of roles from a global perspective (Desai et al., 2005). Protocol rules specify how interaction messaging events trigger operations on commitments (Desai et al., 2009) and specify data dependencies between messages. The rule-base nature of commitment protocols facilitates adaptation via protocol composition (Desai et al., 2006). Given a protocol, behavioral skeletons for interacting roles can be generated. Role skeletons can then be augmented by local business policies to obtain executable business process for each role.

Commitments-based approaches clearly distinguish the global view of an interaction, i.e. interaction protocol, from the local views of interacting roles. The global view is projected onto each role to obtain a local view in a systematic manner (Desai et al., 2005). Protocol composition provides flexibility for disciplined adaptation. An interaction may be modularized into reusable fine-grained protocols that are composed together in an iterative way to form complex protocols.

Alternate flows can also be incorporated into local views as events (Chakravarty & Singh, 2008). Protocol specification and composition have formal groundings as well as a practical basis for implementation (Desai et al., 2005) and (Desai & Singh, 2007).

Even though commitments elevate the representation of inter-connections from messaging to business concepts, their relation to stakeholder goals has not been investigated. Commitments-based approaches assume that interaction design starts with constructing a protocol. Rules governing the sequencing of messaging are considered a matter of convention (Desai et al., 2009) rather than an artifact derivable from the requirements motivating an interaction. By supporting only outside-in development (from global-to-local), these approaches de-emphasize local views. The possibility that adaptation to an interaction may be driven by changes in local models is not considered. Furthermore, there is no provision for determining the impact of local business policies on the sequence of protocol messages.

3.1.1.5 Business Goals and Interconnections

While explicit specification of stakeholder goals enables requirements-driven adaptation of participant local views, explicit specification of participant interconnections enables reasoning about participant inter-dependencies for fulfilling their goals. Explicit specification of both goals and interconnections enables requirements-driven adaptation of a choreographed interaction. Agent-Oriented Methodologies provide notations for capturing business goals of interconnected agents. Several methodologies were developed over the past couple of decades, the most prominent of which are MaSE, GAIA, Tropos, and Prometheus (Dam & Winikoff, 2004). Although these methodologies share many notational commonalities (Padgham et al., 2008), Tropos (Bresciani et al., 2004) excels in providing the most explicit representation of agent inter-dependencies for fulfilling business goals (Lamsweerde, 2004). In addition to supporting the representation of goals of each agent (i.e. participant) and their refinement, Tropos supports the representation of goal delegation between agents via "dependencies". Explicit representation of dependencies facilitates reasoning about how an interaction between a set of participants is motivated by the need to fulfill their goals (i.e. via delegation). Tropos offers a range of modeling and analysis activities that support all phases of software development from early

requirements analysis to detailed design and implementation (Castro et al., 2002). The methodology offers a framework for outlining system context, identifying roles, their goals and dependencies, refining goals into sub-goals and eventually into activities. Dependencies between roles are used to identify required capabilities of each participant in an interaction (Penserini et al., 2006).

This wide lifecycle coverage of Tropos was extended to cover service-based implementation where capabilities were used to derive service interfaces to be implemented by each participant (Lau & Mylopoulos, 2004). The identified interfaces are static, that is, they do not specify an interaction protocol. The detailed design and implementation phases of Tropos were later extended to support service interactions (Penserini et al., 2007). During detailed design local workflows of each participants are described using UML activity diagrams while message exchanges are specified using UML sequence diagrams. In the implementation phase executable agent behavior is generated from detailed design artifacts. Although these extensions provide systematic means for obtaining an executable agent interaction from requirements, there are two shortcomings. First, no guidance is provided for composing agent behavior to obtain a global interaction protocol. Second, operational sequence diagrams are not derived from requirements but rather treated as an independent design artifact. Thus, systematic requirements-driven adaptation of interaction specification is not supported.

Koliadis et al. (2006b) propose a systematic approach for relating requirements, expressed in a Tropos model, to interaction specification in BPMN. BPMN activities are annotated with their immediate effects, i.e. post-conditions, using a constrained natural language. Effect annotations are accumulated by systematically traversing a BPMN specification, thereby obtaining a behavioral specification of every activity (Hinge et al., 2009). Consistency between a BPMN specification and the corresponding Tropos model is checked using a set of rules (Koliadis et al., 2006a). Using these rules along with effect annotations it can be determined whether the specification of BPMN activities fulfills goals specified in a Tropos model. Changes in a Tropos model are systematically propagated to the corresponding BPMN specification using another set of rules thereby maintaining consistency between them (Koliadis et al., 2006b). This approach

adequately addresses the need to specify local stakeholder requirements, capture dependencies that motivate an interaction, and relate requirements models to operational ones. The approach can benefit from extensions at both ends of the abstraction spectrum. First, even though guidelines are provided for obtaining a BPMN model from an adapted Tropos model, no detailed guidance is provided on evolving a Tropos model to accommodate emergent requirements. Second, the approach does not provide an automated way for obtaining a choreographed messaging specification from either a Tropos models or a BPMN model.

3.1.2 Integrating Services and Requirements

Moving from requirements to interaction specification, i.e. deriving the latter from the former, is but one way to obtain service interactions that satisfy stakeholder requirements. It may be argued that these top-down approaches entail the overhead of maintaining consistency between two disparate representations. It may also be argued that tightly coupling representation of requirements specification with that of a messaging specification allows systematic "co-evolution" of both specifications. By integrating requirements with corresponding interaction specifications the latter can be evolved by adapting requirements without the need for derivation. This integration is achievable by either using requirements as annotations to interaction specifications, or by using the same representational constructs for requirements and interaction specifications to combine them into a single representation.

3.1.2.1 Using Requirements as Annotations of Interaction Specification

Identifying and representing variability is a well-investigated technique in the context of domain analysis (Prieto-Diaz, 1990). Although "aspects" are mostly known for capturing cross-cutting concerns in software artifacts (Kiczales, 1997), aspects can as well be used to represent points of variability in a specification. Charfi & Mezini (2004a) apply the concept of "adaptation via points of variability" (Curbera et al., 2005) to process descriptions. To infuse flexibility into a business process, business rules are integrated into a BPEL specification using an aspect-oriented language called AO4BPEL (Charfi & Mezini, 2004b). This integration introduces control points at which a process flow may be adapted by changing business rules. However, this approach does not elevate the nature of the flow-oriented process to the level of business goals, as

AO4BPEL rules are detached from the goals that motivated their existence. Requirements-driven adaptation of flow-oriented rules can be as hard as that of the BPEL specification in the first place. In other words, AO4BPEL aspects encode varying characteristics of design rather than relations of variations to their causes (Liaskos et al., 2006). This approach is thus useful for modularizing computations in a BPEL specification, rather than using requirements to reason about adapting a business process.

A framework through which organizational requirements in Tropos are integrated into a BPEL business process specification is proposed in (Kazhamiakin et al., 2004). Temporal logic is used to specify behavior of Tropos model elements, e.g. activities. BPEL elements are annotated with the temporal logic formulas of corresponding Tropos elements. By using a model checker, a BPEL process is verified to satisfy requirements in the corresponding Tropos model. For instance, it is possible to determine that an activity specified in a Tropos model has been fulfilled when the corresponding BPEL element has completed execution. This approach enables verifying compliance between requirements and BPEL specifications, i.e. it only deals with participant local views, it does not provide guidance on adapting the global view of an interaction.

3.1.2.2 Combining Requirements and Interaction Specification

In contrast with using business rules as annotations in AO4BPEL, business rules have also been used to combine requirements and interaction specification into a single representation. A framework for integrating business objects, scenarios, business processes, and services is proposed in (Mazzoleni & Srivastava, 2008). The framework uses business rules to encode facts about all these entities, structural relations between them, and required impact propagation between them in the event of change. Given an emergent change in a business object or a business process, expressed as a rule, an inference engine determines all other entities impacted by the change. The framework has some limited practical application, but its utility is bounded since it only helps determine *what* entities may need to be adapted but does not provide any guidance on *how* to adapt them.

Another rule-based framework for integrating requirements and service interaction specification is proposed in (Orriëns & Yang, 2006). The framework captures an interaction specification at three levels of abstraction, namely: strategic, operational, and implementation. All elements involved in the specification of an interaction at all levels of abstraction (roles, resources, activities, events, messages, services, etc.) are represented using a few primitives: entity, attribute, link, and mapping. A high degree of flexibility is attained by representing all structural and behavioral aspects of an interaction as rules. However, constructing and maintaining a set of rules that describe an interaction to any level of detail is left to be an overwhelming manual task.

3.1.3 From Services to Requirements

Some of the methods that provide a path from requirements to service interaction specification also provide a path from service interaction back to requirements. The motivation of such approaches is to enable identifying and performing changes at an operational level, and not only at a strategic level. Although our target is requirements-driven adaptation of service interactions, we cover approaches that proceed in this "reverse" manner, from service interaction to requirements, for completeness. Representative of these approaches is the approach that relates Tropos models to BPMN specifications, previously presented in section 3.1.1.5 (Koliadis et al., 2006a). In addition to guiding adaptation of a BPMN specification in response to changes in the corresponding Tropos model, guidance is also provided for adapting a Tropos model when the BPMN specification changes. This reverse engineering of requirements models may come in useful in situations where operational specification for an interaction already exists whereas only a partial requirements model is available.

This line of reasoning is taken a step further in (Jureta et al., 2007) by arguing that, due to the openness of service-oriented systems, it is not feasible to fully specify requirements upfront. Characteristics of interacting services may change over time in ways that were not predictable at analysis and design time. "Client Requirements", that specify quality of service expected by service clients, are allowed to vary dynamically. At runtime, each service request specifies constraints on desired output including quality criteria for evaluating service output.

Requirements specifications are updated at runtime to reflect characteristics of service requests thereby keeping requirements consistent with the expectations of service clients. This approach allows for a high degree of flexibility in specifying desired service qualities, but it deemphasizes the importance of changes that arise in service orchestrations (i.e. local view) as well as service coordination (i.e. global view). Requirements specifications for composition and coordination play the role of system documentation rather than a driver for a service interaction design.

Similarly, it has been argued that it is important to update requirements dynamically to reflect information learned about available services. Gehlert et al. (2008) envision an approach where a requirements engineer specifies initial requirements using goal models, and refines goals into activities. The refined models are compared with services in a service registry to find matching services, a step which aligns existing services with requirements. A requirements engineer uses matching services to enhance and augment initial requirements, thereby aligning requirements to existing services. This approach assumes that services are annotated with goal models representing a single local view, so it only addresses service orchestration.

3.2 Representing and Relating Interaction Specification Viewpoints

Choreography addresses the needs of a global stakeholder, e.g. regulatory agency, interested in monitoring an interaction. Participants in an interaction, on the other hand, participate in an interaction to satisfy needs relevant from their local point of view. Distinguishing between local and global views is essential for serving needs of all stakeholders, and relating views together is necessary for maintaining consistency between them.

The distinction between behavioral interfaces, of components or services, and coordination logic that connects these interfaces together (i.e. choreography) has long been made. Connectors (Allen & Garlan, 1997) were proposed as glue that bonds component interfaces together. Connector specification was formalized using CSP (Hoare, 1985) thereby enabling automated checking for compatibility between component interfaces. Adaptation was envisioned via connector composition (Lopes et al., 2001) but many non-trivial research questions arise (Garlan, 1998). Connectors have traditionally specified inter-connection and interaction between software

components in a machine-oriented form. The lack of an accompanying business-level representation leaves the challenges of requirements-driven adaptation of an interaction unanswered. In a service-oriented context the adaptation problem is more complex due to the autonomy of participants as well as potentially de-centralized evolution.

Agent-oriented approaches provide frameworks for representing goals and activities of autonomous interacting participants. The i* framework (Yu, 1995), in particular, distinguishes between "Strategic Dependency" models, that capture a global view of agent inter-dependencies, and "Strategic Rationale" models that embody local goal models of each agent. Tropos extends the i* framework and provides some explication of relations between views (Traverso et al., 2004). Agent local views are related via dependencies on each other for goal fulfillment, thereby enabling change propagation between them (Koliadis et al., 2006a).

Even though Tropos provides a suitable framework for capturing the global and local views and relating them, we inquired into other approaches that relate the global to local views. These approaches can be categorized into four categories which we discuss in detail:

1. Global view as the primary specification: The global view is considered to be the primary specification of an interaction. Local views are obtained from the global model via automated generation,

2. Local views as the primary specification: Participants local views are considered to be the primary specification of an interaction. The global view is obtained via composing local views,

3. Global and local views as independent specifications: The global and local views are considered to be complementary specifications of an interaction. The global model and local models are created separately and checked for consistency, and

4. Global and local views in an integrated specification: The global and local views are specified using the same set of representational constructs, thereby enabling their combination into the same representation.

3.2.1 Generating Local Views from Global View

This class of approaches assumes that the global model of an interaction is completely specified prior to specifying local models, and hence, advocate automatic derivation of local models from the global model. Mendling & Hafner (2005) present a generative approach which takes advantage of the XML-based nature of Web service interaction standards. WS-CDL is used to specify an interaction protocol that represents the global model. BPEL local models for interaction participants are automatically obtained by applying an XSLT transformation (W3C-XSL, 2007) to a WS-CDL document. The transformation is purely syntactical and its correctness is left unproven. The informality and syntax ambiguity of WS-CDL (Barros et al., 2005a) is contrasted by the precision and formalism of process calculi. Zhao et al. (2006) put forward a transformation to project the global view expressed in a process calculus to obtain participant local views. Correctness of the transform is proven and it guarantees consistent participant behavior under both normal and exceptional conditions (Cai et al., 2009). However, business-level specification of an interaction is not considered.

"Let's dance" (Zaha et al., 2006a) is a language that employs a graphical notation for specifying a service-oriented interaction. The graphical notation is intended to be comprehensible by analysts and architects participating in interaction design. An interaction is specified as a conjunction of constraints on pairs of messages. Precedence constraints are used to specify partial ordering of messages, and inhibition constraints specify mutual exclusivity between a pair of message sending events. An interaction is described from a global point of view, from which local views are derived. Local views can then be translated into executable BPEL descriptions (Zaha et al., 2008). The semantics of the language were formalized using PI calculus and it was proven that an interaction described from a global point of view can be realized when projected to local views of interacting roles (Decker et al., 2006b). The representation is limited to messaging activities and physical activities are not addressed.

Physical activities are as much an integral part of an interaction as is electronic messaging, and workflow languages allow incorporation of physical activities in the specification of service-oriented interactions. Representative of these approaches is that presented by van der Aalst

(2004) for designing inter-organizational workflows. A public workflow (i.e. choreography) is represented using a variant of Petri Nets that specifies executed tasks (i.e. message sending or physical activities) and causal relations between them. Tasks in a public workflow are partitioned into a set of domains (i.e. local views) and a private workflow (i.e. orchestration) is generated for each domain. Generated private workflows are consistent with the public workflow and the public workflow is guaranteed to be deadlock-free. No guidance is provided by this approach on how to incorporate into the private workflow business activities beyond those present in the public workflow.

Although this class of approaches allows systematic derivation of local views that are guaranteed to be consistent with a choreography specification, they only support "outside-in" development. That is, autonomy of participants is de-emphasized and thus these approaches are not suitable for reasoning about changes that arise in local views or propagating these changes to the global view.

3.2.2 Generating Global View from Local Views

A choreographed interaction is established between participants that may already have functioning processes (Decker & von Riegen, 2007). To accommodate this scenario, this class of approaches attempts to build a composite (i.e. global) view of an interaction given an established set of local views. Representative of these approaches is the Petri-Nets-based approach proposed by Martens (2005). The local view of each participant is captured in a "workflow module". A workflow module is a Petri Net with a set of input and output places designated as the public interface of the local view. The notion of syntactic and semantic compatibility between workflow modules is formalized and a technique for building a composite workflow module is proposed. Although the suggested composition is systematic, a global view provided as a workflow module does not distinguish between globally observable messaging and local participants workflows. Similarly, Dijkman & Dumas (2004) use Petri Nets to represent interfaces exposed by interacting roles, and a specification of choreography can be derived from the collective behavior specified by these interfaces. These approaches are useful for determining whether or not it is possible to

establish a choreographed interaction between a set of local views. However, they do not support business-level reasoning about adaptation.

It has been further argued that the specification of a choreographed interaction should be derived from the actual execution of running processes. Workflow mining addresses this need by reversing the design process (van der Aalst et al., 2003b). Process execution logs are analyzed to infer rules that govern execution. An algorithm is proposed in (van der Aalst & Weijters, 2005) to obtain a specification of choreographed behavior of a set of processes. Mining approaches may be useful in discovering the control flow rules in the absence of explicit process specification. However, contextual rules, e.g. those globally imposed by a regulatory agency, are not distinguishable from local business policies of participants. It is thus hard to reason about global business rules or their adaptation. This is further complicated since there is no guarantee that the analyzed processes are representative of their business domain. It is thus not possible to generalize extracted rules governing a set of processes to a wider context.

3.2.3 Using Separate Representations for the Global and Local Views

Allowing local views to evolve independently from the global view, and vice versa, requires means for checking their consistency. A representative of consistency-checking approaches is that presented by Foster et al. (2006) where an interaction is specified using Message Sequence Charts (MSC), the global obligations are represented using WS-CDL, and the local views are represented as BPEL processes. The local views are then combined into a composite view which is then converted into a Labeled Transition System (LTS). The WS-CDL representation is also converted into an LTS. The LTS representation of each of the global and the composite model are formally checked for consistency against the MSC specification. Similarly, Busi et al. (2005) formalize the notion of conformance between choreography, represented using WS-CDL and a set of orchestrated processes, represented in BPEL. Both representations are converted to an LTS and checked for consistency using bisimulation. A similar approach is presented by Baldoni et al. (2005), albeit with the limitation of dealing with interaction between two roles at a time.

This class of approaches enables automated verification of implementation vs. messaging design specifications, in the form of MSC. They also facilitate the verification of consistency

between a set of BPEL processes and a WS-CDL choreography that specifies their collective observable behavior. MSC message specifications are far detached from business requirements that drove the design, thereby leaving requirements-driven evolution unaddressed. Furthermore, MSC have expressive limitations regarding concurrency, synchronization, and representing alternate scenarios (van der Aalst, 2004). To overcome the limitations of MSC, Petri Nets have been adopted in other checking approaches for representation of an interaction. In addition to choreography and orchestration two additional views may be separated: "interface behavior", which represents an interface exposed by one role to another, and "provider behavior" to represent collective observable behavior of a role with all other interacting roles. The four views have been represented using Petri Nets and related together to enable formal consistency checking (Dijkman & Dumas, 2004).

Checking approaches do not support step-wise adaptation as they do not facilitate making incremental changes to behavioral specification. Furthermore, these approaches do not provide guidance on how to evolve one representation to match the other if inconsistencies are found. Rindele et al. (2006) propose an approach to guide the evolution of one view given a change in another by enabling propagating changes back and forth between the global and local views. Observable behavior of each role is represented as a Finite State Automaton (FSA) while local views are represented in BPEL. The BPEL syntax is then translated to an automaton and a mapping is established between it and the FSA of the behavioral interface. A change to add/remove a message is propagated using the established mapping from the local view to the global view and vice versa (Wombacher, 2009). Although two-way change propagation is made possible, no attention is paid to the business meaning of changes. That is, there is no assurance that the changes to the local views will satisfy the goals they are supposed to achieve.

3.2.4 Combining the Global and Local Views

Whereas using two disparate representations for the global and local views requires techniques for keeping them consistent, generating one from the other makes it hard to adapt the generated view. Approaches that combine all views into a single representation, attempt to strike a balance between guaranteeing consistency and ease of adaptation. These approaches use the

same constructs to represent both types of views, thereby allowing combining views in one model. When desired, the global and local views can be obtained as projections of the common model.

Wieczorek et al. (2009) propose an approach that uses a common metamodel, based on a state transition system, for the global and local views of message exchange. Using the same representational constructs for both types of views eases change propagation between them. Three projections of message exchange are considered: "Send", "Receive", and "Observe" that correspond to the sequence of message sending events, message receiving events, and message observation events on a communication channel, respectively. Each projection serves a different need. For example, the "Receive" projection takes into account out-of-order delivery of messages. The approach has utility in generating tests and verifying the compliance of local implementations to the global model under varying characteristics of a communication channel. Otherwise, the approach has many limitations, the foremost of which is that it is only applicable to two interacting roles. Additionally, even though an interaction is specified at an operational level, the approach provides no support for generating an implementation, which has to be done manually.

To circumvent the need to derive one view from another or even derive an implementation from requirements, Orriëns & Yang (2006) propose a common rule-based representation for all views of an interaction at all levels of abstraction. Specifications of all interaction views and at all levels of abstraction are captured using a handful of elements drawn from a common metamodel. Although flexibility is attained by representing all aspects of an interaction as rules, a flat rule-based representation makes it not suitable for business-level reasoning about adaptation. It is unclear how to attribute a rule to a stakeholder's need or determine the impact of a change in one rule on other stakeholders' needs.

3.3 Providing Guidance for Disciplined Adaptation

Providing guidance to support service-oriented development and adaptation is one of the grand challenges of service-oriented systems engineering (Papazoglou et al., 2005). The most

basic guidance is to enumerate potential adaptation operations applicable to certain metamodels of interaction specification. Each adaptation operation encapsulates guidance on how to apply it to an interaction specification. Multiple adaptation operations may be applicable in a given situation, and thus guidance is also required for making choices between potential alternatives. After applying an adaptation operation to an element of an interaction specification, it may be necessary to apply further adaptation operations to other elements to keep different parts of the specification consistent with each other. Guidance is thus required for driving the adaptation process by propagating changes to all affected parts of an interaction specification. Even though most approaches discussed so far provide some form of guidance, we discuss in this section research efforts whose significant part of their focus is on:

1. Categorization of changes and providing catalogues of adaptation operations.
2. Evaluation and selection from among alternatives ways for performing an adaptation.
3. Automating the progression of the adaptation process.

3.3.1 Categorizing Types of Change and Adaptation Operations

During step-wise adaptation of an interaction specification, adaptation operations are performed in each step. An adaptation operation transforms one valid interaction representation to another. Providing a catalogue of adaptation operations applicable to a representation of an interaction is an essential part of guidance. Similar to patterns catalogues (Gamma et al., 1994) a catalogue of adaptation operations compiles knowledge about each operation such as: situations when it is applicable, considerations when performing it, its consequences, and variations. We review catalogues of adaptation operations categorized by artifacts targeted for adaptation: specification of service interfaces, specification of participant behavior, and specification of interaction between participants.

3.3.1.1 Operations for Adapting Service Interfaces

Developing guidance on how to reconcile mismatches between component interfaces has long been a hard problem (Garlan et al., 1995). SOC provides a solution to mismatch in language and

syntax via XML-based standards such as SOAP and WSDL (see section 2.1.1). And although attempts to standardize on semantic descriptions, e.g. based on the Web Ontology Language (OWL) and its variant for services (OWL-S), have not yet gained a lot of acceptance, some success has been achieved in semantic-level adaptation. The Nile system (Trastour et al., 2003) uses a predecessor of OWL to bestow conceptual semantics on XML-based representations of service interface specification. The technique was demonstrated in the limited context of RosettaNet[1]. SOC, however, does not provide a general solution for resolving mismatches in operation signatures.

Benatallah et al. (2005) give a classification of operation signature and service-interface-protocol mismatches. Signature mismatches cover cases where the expected and actual service interfaces possess different operation signatures. Protocol mismatches capture cases where the expected and actual service interfaces disagree on message ordering, extra/missing messages, and message merge/split. To resolve these mismatches, service interface adaptor templates are proposed as canned process fragments. Each template is instantiated to resolve a corresponding mismatch when identified, manually, by an architect. A semi-automated technique for identifying service interface mismatches was later introduced (Nezhad et al., 2007). An automated tool analyses service interfaces, presents an architect with mismatches, including potential deadlocks, and provides help on resolving them.

In addition to tackling message-ordering mismatches, Hiel & Weigand (2009) tackle mismatches in the structure of XML messages and constraints on message content. Potential mismatches are classified into harmless, solvable, and problematic. Operations for adding, removing, and changing parts of message structure are encoded as edit operations. A sequence of edit operations forms an "edit script", which when applied to a service interface produces an adapted version of the interface. For solvable mismatches, an edit script that produces one interface from another is generated automatically by comparing two service interfaces. This approach is useful for adapting message formats in WSDL descriptions.

---

[1] www.rosettanet.org

Utility of this class of approaches is limited to guiding primitive adaptation to operational service interface specifications, and only from the point of view of a service client. They do not support business-level reasoning about adaptation nor do they support multi-participant interactions where each autonomous participant has a separate local view.

3.3.1.2 Operations for Adapting Participant Behavior

Each participant in an interaction is an autonomous entity that owns a local process. Each local process may evolve independently from the global interaction specification. Kongdenfha et al. (2006) propose a set of mismatch templates for adapting BPEL processes. It is argued that "adaptation logic" can be separated from business logic and that adaptation can be treated as a cross-cutting concern using aspects. Each template contains a set of "advice" that define adaptation logic as a process snippet along with "pointcuts" that specify where adaptation of each advice may be applied. Pointcuts identify points in the XML description of a BPEL process along with conditions under which the corresponding adaptation applies. Once an architect identifies a mismatch a tool helps with the generation of adaptation logic from the corresponding template and integrates the logic into a process (Kongdenfha et al., 2009). However, identifying mismatches is labor-intensive since an architect has to manually go through the textual specification of a BPEL process.

The approach presented by Weber et al. (2007) abstracts away from textual process specification and addresses adaptation of abstract process structure. The approach uses patterns to classify structural changes that a business process may undergo. These patterns codify changes to process structure such as inserting, removing, replacing, or extracting a process fragment. Each pattern encapsulates guidance in the form of considerations when applying the change codified by the pattern. It is argued that these patterns provide a more modular approach to adaptation than finer-grained manipulations of nodes and edges in a process flow, thereby making the adaptation process less error-prone (Reichert et al., 2009). The categorization of changes and the itemization of design considerations associated with each type of change can potentially be useful when making individual changes to a local view. However, changes to a

process do not take into consideration their effect on observable behavior of a participant, and no means are provided for change propagation to the global view.

Realizing the need to maintain interoperability between interaction participants while catering for the autonomous nature of participants brought about the approach proposed by Baldoni et al. (2009). Participant behavior is specified as a labeled state machine. Each state is annotated according to whether a participant is leading (i.e. making a choice in the process) or following (i.e. responding to choices made by other participants). Interoperability between participant state machines is formally defined based on the notions of leading and following, which abstracts from message sending and receiving. Types of changes to adapt participant behavior are codified as "edit operations". Performing an edit operation on a state machine of a participant guarantees that their local adapted behavior remains conformant with their observable behavior. Although these operations guarantee correctness of the resulting adapted specification, the approach does not provide guidance on which operation to apply in a given situation and does not support local changes that require propagation to the global view.

3.3.1.3 Operations for Adapting Service Interactions

Adapting inter-dependencies between interaction participants and inter-participant messaging are central to adapting choreographed interactions, and so are catalogues of relevant adaptation operations. Barros et al. (2005b) propose a catalogue of inter-participant messaging patterns. The catalogue classifies messaging interactions according to number of participants, number of exchanged messages, and whether messaging is intermediated. The patterns were intended as a reference against which features of interaction specification languages, e.g. WS-CDL, are assessed. The behavioral semantics of each pattern as well as compositions of patterns were formalized (Barros & Boerger, 2005). Formalization of pattern compositions potentially enables adapting a messaging interaction via composing patterns. However, the patterns are purely operational in nature and the approach does not support imparting them with business meaning.

As discussed in section 3.1.1.4 "commitments" provide a business-level representation of participant inter-connections. Singh et al. (2009) introduce commitment-based patterns for specifying participants inter-connections. Patterns such as "Revert Offer", "Penalize", and

"Transfer Responsibility" abstract away from messaging specification, thereby providing flexibility in specifying an interaction. Each pattern is expressed as a statechart that relates business events to commitment lifecycle transitions, such as "satisfy", "delegate", and "dismiss". The patterns were experimented with in specifying a few commercial cases (Telang & Singh, 2010) but have not been tested in an adaptation scenario. The task of building a complete set of patterns with respect to the commitment-based metamodel has not been discussed.

Based on the i* metamodel, a complete set of adaptation patterns is presented by Krishna et al. (2009). The patterns are part of a technique for maintaining consistency between requirements models, in i*, and behavioral specification in the Z language. An i* model is represented as a Z schema and additionally every element in the model is represented as a Z schema of its own. A two-way mapping is laid out between an i* model and the Z language. Sixteen categories of changes to an i* model are identified: addition/deletion of Dependencies, Activities, Goals, Resources, Softgoals, Means-end links, Activity-decomposition links and Actors (Krishna et al., 2004). Rules are worked out for reflecting each category of change in the corresponding Z schema of affected model elements. The approach allows refinement of a Z description with further information beyond what is represented in the i* model, such as adding dependency sequencing information. The resulting refined Z schema can still be adapted to reflect changes in the original model. Also, changes in the refined schema can be mapped back to the corresponding i* model thereby enabling bidirectional change propagation. Although, the approach provides guidance on adapting a formal representation coupled with an i* model, it does not provide guidance on how to update the i* model itself to accommodate an emergent requirement.

3.3.2 Facilitating Evaluation of Alternatives

More than one adaptation operation may be suitable for incorporating a given emergent requirement into an interaction specification. Moreover, applying an operation may involve making choices among alternative ways of applying it. For a non-trivial interaction, the space of alternative ways to perform a given adaptation can be large. Providing systematic means for exploring this space and evaluating the suitability of each alternative is thus essential for

facilitating adaptation. As discussed in section 2.4.4.1, supporting runtime adaptability is an open research area which is outside the scope of this thesis. However, for completeness, we review approaches that facilitate exploration and evaluation of alternatives both at design-time and runtime.

3.3.2.1 Design-time Evaluation

At design-time, an interaction specification is adapted by applying adaptation operations to some model of the interaction. An adapted model is compared to other models resulting from alternative adaptations. Qualitative and/or quantitative techniques are required to aid with this comparison. One such approach for adapting an i* specification of an interaction uses effect annotations proposed by Koliadis et al. (2006a) and discussed in section 3.1.1.5. The notion of "equilibrium" between a given i* model and a corresponding BPMN process is defined. The two models are in equilibrium if all activities in the i* model are represented in the process model and all goals in the i* can be fulfilled by at least one path in the process. A proximity relation between models is defined to assess how similar two models are. When a change in a model perturbs equilibrium, the proximity relation is used to help assess proposed adaptations and select the alternative that involves minimal changes (Ghose & Koliadis, 2008). The approach assumes the existence of a library of process fragments and proximity is measured with no attention to business meaning of changes.

Relating a change to the business goals is meant to achieve is crucial to ensuring that the change satisfies these goals. Giorgini et al. (2003) present a framework that enables reasoning about options for goal "satisfiability". Goal satisfiability is defined as the degree to which a goal is satisfied on a scale from zero to one. In addition to AND-OR decomposition of goals, a goal graph captures positive and negative contribution links between goals. Precise semantics are given to goal links, both in qualitative and quantitative forms. An algorithm for propagating contributions throughout a goal graph is laid out. Using this algorithm, a measure of satisfiability of a goal can be computed given an assignment of satisfiability for the rest of the goals in a graph. The framework is useful for systematically evaluating alternative goal refinements given satisfiability assignment. However, it does not help find alternative ways to assign satisfiability

to goals in order to fulfill a given goal. That is, it works bottom-up but not top-down. Furthermore, it only supports reasoning about goals from a single point of view.

A choreographed interaction involves reasoning about goals from multiple points of views. Bryl et al. (2006) propose a framework for exploring and evaluating alternative goal assignment and refinement from multiple points of view. The evaluation process starts with a set of actors, a set of initial goals organized in a goal graph, and a set of goal-to-actor assignments. An AI planning tool is used to generate different assignments which lead to goal achievement. For each goal, an actor may choose to fulfill the goal locally, if they have the capability to do so, or choose to delegate it to another actor. Each alternative is evaluated using metrics derived from game theory (Osborne & Rubinstein, 1994). The approach selects an alternative that achieves a state of equilibrium among all actors, i.e., a state where the actors are contented with their goal assignments. The approach can be a useful negotiation tool between participants at early stages of specifying an interaction where goal assignment is a key concern.

3.3.2.2 Runtime Evaluation

Deferring evaluation of alternatives to runtime allows more flexibility in choosing between adaptation alternatives. An interaction may then be adapted dynamically in response to changes in execution conditions. Agent-oriented methodologies (see section 2.3.2) provide a foundation for this flexibility via autonomous agents that are capable of adapting dynamically to their environment. An agent-oriented approach for incorporating alternatives and logic to choose between them into executable agent behavior is proposed by Penserini et al. (2007). At design time, agent capabilities are encoded as activity diagrams which are translated into state machines. At runtime, an agent responds to runtime events and consults its current state as well as the state of the environment to select between capabilities. An agent chooses to activate capabilities that would achieve its goals at the time of making a choice. Although this approach infuses some flexibility into interaction specification, all alternative behaviors have to be identified and specified at design time.

In addition to providing flexibility, runtime evaluation of alternatives can be used as a mechanism for handling service failures. He et al. (2008) present a technique for evaluating

alternatives for recovering or replacing a failed service. It is assumed that a certain cost is associated with bringing about each alternative and a certain value is realized by bringing it about. The value and the cost are computed to estimate the expected "profit" of each alternative. A probabilistic model for computing the value of each alternative is adopted. Formulas are worked out for computing cost associated with a composition of services via sequence, parallel split, merge, choice, and other control flow constructs (van der Aalst et al., 2003a). For successful runtime replacement of services, the approach assumes that the services semantics are sufficiently elaborated in a shared registry. Whereas the approach may have utility for evaluating QoS characteristics of alternatives from an operational service client point of view, requirements-driven adaptation of multi-viewpoint interactions is outside its scope.

As discussed in section 2.4.4.2 several process instances may already be in the middle of execution when the corresponding process specification is adapted. ADEPT2 is a framework for evaluating alternatives for adapting running process instances to conform to a new process specification (Reichert et al., 2009). Labeled Place-Transition Nets are used to represent control flow of a process. A set of criteria is put forward for evaluating structural and behavioral correctness of a process instance after dynamic adaptation. Although, ADEPT2 ensures compliance of adapted instances with the new process model as well as freedom of deadlocks, it does not tackle interaction of a process with other autonomous processes.

3.3.3 Guiding and Automating the Adaptation Process

Adaptation operations are gadgets for adapting an interaction specification. To achieve a desired adaptation, it is typically necessary to apply several operations. Without guidance on how and when to apply these gadgets an architect is left with trial and error based on intuition. Methodologies and techniques for combining adaptation operations are thus called for. These techniques typically provide guidance to perform macro or micro changes. Macro changes involve successively composing specification fragments to achieve a desired adaptation. Micro change involve stepwise incremental application of adaptation operations to a specification until no further change is required.

3.3.3.1 Adaptation via Service Composition

Availability of standards-based service interface descriptions enabled emergence of many techniques for automated service composition via AI planning (Rao & Su, 2004). A representative of these approaches is proposed by Ponnekanti & Fox (2002), which automates composition of Web services to achieve a desired effect. Given a repository of component services annotated with pre- and post-conditions and a desired final state, a tool produces a plan for obtaining the desired state by combining component services. These approaches typically assume that execution of component services does not have side-effects, which does not fit the case of stateful multi-participants interactions.

Similarly, availability of a repository of composition elements enables rule-based adaptation via composition. Composition elements include activities, flows, events, conditions, as well as rules governing compositions thereof (Orriëns et al., 2003a). An architect interacts with an automated composition assistant to specify a composition out of elements stored in a repository. The desired result of composition is specified through rules that constrain structure and behavior of the composition (Orriëns et al., 2003b; Orriëns & Yang, 2006). Use of a centralized repository assumes single-stakeholder ownership and an orchestrated interaction rather than a peer-based one.

Benatallah et al. (2002) present a peer-based composition approach to adaptation where each peer exposes a service whose behavior is controlled via a "coordinator". Behavior of a coordinator is specified as a statechart annotated with post-conditions. A coordinator controls state transitions of the associated service, according to its statechart, and notifies other coordinators about its state transitions. Adaptation is achieved by generating state charts of a composite service from those of individual services. Behavior of coordinators can be configured dynamically with generated state charts. Similar to other composition approaches, this approach relies on the existence of a repository or a service community (Benatallah et al., 2002) that aggregates service offers with a unified interface.

3.3.3.2 Adaptation via Step-wise Changes

As opposed to composing fragments to achieve a desired adaptation, "step-wise" approaches prescribe steps for incremental adaptation of an interaction specification. These approaches apply an adaptation operation to an interaction specification, and use change propagation techniques to identify further operations to be applied. Amoeba (Desai et al., 2009) is a methodology for specifying and evolving multi-participant interactions based on business protocols. A business protocol specifies an interaction among the participants via commitments (see section 3.1.1.4). The methodology provides step-wise guidance on specifying and adapting an interaction specification. Guidance includes techniques for identifying roles and their interactions, capturing contractual relations via commitments, specifying relations between messages and commitment lifecycle, specifying constraints on message ordering, and composing business protocols. The proposed steps provide a systematic way for propagating changes from the global view to local views, but not the other way around.

Dam et al. (2006) provide guidance for bidirectional change propagation between the global and local views of an interaction, based on an agent-oriented methodology called Prometheus (Padgham & Winikoff, 2004). The approach proposes a UML metamodel for all entities and relations that Prometheus uses to specify an agent-oriented system. The metamodel is annotated with well-formedness rules that constrain the structure of valid models. Adaptation operations for adding to, or removing elements from a model trigger "evolution events" which in turn trigger "evolution action plans". An evolution action plan checks for constraint violation, and in case a violation is identified, further evolution events are generated to restore model validity. The main benefit of this approach is maintaining structural consistency of a model during adaptation, albeit with primitive tool support (Padgham et al., 2005). Although interaction protocols are part of the Prometheus metamodel, no specific action plans are laid out to handle their adaptation in response to changes in business needs. As such, derivation of a multi-participant interaction protocol from goals and activities is not dealt with.

3.4 Gaps Identified with Existing Approaches

Our review revealed two potentially viable paths to tackle the three challenges facing requirements-driven adaptation of choreographed interactions: one path utilizes the Tropos methodology and the other is based on commitment protocols. Both alternatives capture participant inter-dependencies at the level of business requirements, distinguish the global and local views and provide means for relating them, and offer some basis for guiding interaction adaptation. However, commitments-based approaches have one severe limitation which is the assumption of an outside-in mode of interaction specification, i.e. local models are only specified after the global model has been specified. Granting local participant requirements second-class citizenship makes it hard to drive adaptation using emergent local requirements. Furthermore, linkage between commitments and business goals was not established except through combining commitments with Tropos (Telang & Singh, 2009) which begs the question why not use Tropos dependencies to capture participants inter-connections instead of commitments in the first place. We thus chose to adopt Tropos as a starting point for tackling the three interaction adaptation challenges. Nevertheless, in our review we have identified gaps to be filled within the Tropos framework itself. We summarize these gaps with respect to each of the three challenges and indicate how we will address them in subsequent chapters.

3.4.1 Bridging Representations across Levels of Abstraction

Tropos offers adequate support for representing stakeholder goals and successively refining them into architectural specifications (Castro et al., 2002). Original refinement techniques were extended to support later design phases (Penserini et al., 2007) including fine-grained specification of participants inter-dependencies (Telang & Singh, 2009). However, the literature is lacking a systematic way for obtaining the specification of a choreographed interaction protocol from refined Tropos models. Instead of treating a messaging protocol as an artifact derivable from refined Tropos models, existing approaches view messaging protocols as a matter of "convention" (Desai et al., 2009) or an artifact that is designed separately from refined requirements and architectural models (Penserini et al., 2007). The relation established between Tropos models and orchestrated process specification (Kazhamiakin et al., 2004) hints that the

operational specification of service interaction may be derived from refined Tropos models, but no formalization is given for relating Tropos to choreographed messaging.

This gap motivated our work in chapter 5 to investigate how requirements, represented in Tropos, motivate choreographed messaging so as to enable derivation of the latter from the former. A few observations are made by Mallya & Singh (2006) regarding how dependencies in Tropos imply commitment protocols. Although these observations are valid and consistent with our own observations, they fall short of providing a full scheme for deriving messaging protocols from requirements models. Our work in chapter 6 formalizes and extensively elaborates these observations and employs them to propose a technique for automated derivation of choreographed messaging from Tropos models. Tropos was extended to support representing precedence between activities (Fuxman et al., 2004), but it does not capture precedence between dependencies as was noted by Krishna et al. (2009). To plug this gap we extend the Tropos metamodel in chapter 5 with annotations that capture precedence constraints between dependencies. We also annotate dependencies by their nature (i.e. physical or informational) in a manner similar to that proposed by Krishna et al. (2009). We thereby make it possible to capture properties of Tropos model elements without resorting to a complementary representation such as the Z language (Krishna et al., 2004).

3.4.2 Representing and Relating Interaction Specification Viewpoints

The distinction between the global and local views of an interaction is well-understood in the context of service-oriented interactions (Dijkman & Dumas, 2004). The bulk of literature relating these views either enables automated generation of one set of views from another or offers whole-sale checking of consistency between them (Foster, 2006). In particular, approaches that consider local views as being projections of the global view (van der Aalst, 2004; Desai et al., 2006; Zaha et al., 2006b) make it hard to support adaptations that are motivated by changes to a local view. None of these approaches support incremental bidirectional change propagation between views. An exception is the technique proposed by Wombacher (2009) which allows two-way propagation of changes, albeit only for messaging specification which does not support business-level reasoning about adaptation.

Agent-oriented methodologies provide frameworks for capturing and relating the views of the interaction at the business-level. The i* framework, and consequently Tropos, from among other agent-oriented methodologies, is the one that explicitly captures agent dependencies on one another for satisfying goals (Lamsweerde, 2004). However, Tropos was criticized for shifting the emphasis from agent interactions to the structure of the "system-to-be" in later design phases (Desai et al., 2009). This motivated us to establish a flavor of Tropos that maintains the focus on interaction specification during later phases, which we detail in chapter 4.

Explicit representation of dependencies enables relating a global view and a set of local views via their inter-dependencies. Techniques that we have reviewed (Traverso et al., 2004; Koliadis et al., 2006b) hint at how this relation may be established, but only in a coarse-grained manner. When adapting a model element, it is necessary to propagate changes to all affected model elements in the same view and in other views to keep views consistent. The need to provide fine-grained relations between views motivated our work in chapter 5 to crystallize separation of views and formalize their relations using dependencies. Whereas commitment protocols capture conditional obligations, Tropos dependencies are unconditional (Telang & Singh, 2009). To remedy this deficiency, we extend Tropos with dependency annotations that capture pre-conditions on dependency fulfillment.

3.4.3 Providing Guidance for Disciplined Adaptation

We surveyed a range of adaptation operation catalogues. The only provably complete catalogue of operations with respect to Tropos models is that detailed by Krishna et al. (2009). We adopt this set of operations as building blocks for our adaptation guidance technique in chapter 7. Given our extension of Tropos to annotate dependencies with conditions, we had to provide operations for adapting conditions on dependencies as well as on other elements in a model.

Adaptation operations must guarantee that an adaptation of a valid model produces another valid model. Agent-oriented approaches, such as Prometheus, employ a metamodel to encode constraints governing construction and validation of interaction models (Dam et al., 2006). On the other hand, the original Tropos metamodel (Giunchiglia et al., 2002) and its extensions (Susi

et al., 2005) fall short of capturing detailed relations of participant local models, as per the focus on the system-to-be in later design phases (Desai et al., 2009). To capture constraints on Tropos models constructed in the flavor described in chapter 4, we formalize an extension of the Tropos metamodel in chapter 7.

We ascertained that exploration of alternatives is well-covered in early development stages dealing with high-levels of abstraction (Bryl et al., 2006). However, we found that exploration of alternatives for adapting Tropos models at later stages is not well-covered. We thus provide guidance on exploring alternatives in our adaptation process. On the other hand, evaluation of alternatives seems to be a fairly well covered topic both during the early Tropos stages (Giorgini et al., 2003) as well as later stages (Fuxman et al., 2004) so we designed our adaptation framework to allow incorporating any off-the-shelf evaluation technique.

We established that existing methodologies for guiding combination of adaptation operations either do not address the adaptation of interaction protocol (Dam et al., 2006), guide propagation protocol changes in an outside-in direction only (Desai et al., 2009), or do not provide detailed guidance on propagating changes between views (Koliadis et al., 2006b). This gap motivated the bulk of our work in chapter 7 to provide detailed guidance on performing adaptation operations and propagating changes between views.

3.5 Chapter Summary

This chapter reviewed research efforts that tackle adaptation of service-oriented interactions. In our review we used three lenses, one for each challenge that faces the adaptation, namely: relating representations of an interaction across levels of abstraction, relating the global and local views of an interaction, and guiding the adaptation process. We identified some gaps in existing research with respect to each challenge and we used these gaps to motivate our work in the coming chapters. We found that the Tropos methodology provides a basis that we can build our contributions upon. We present some details about the methodology and our proposed usage of it in the next chapter.

Chapter 4. Specifying Choreographed Interactions in Tropos

This chapter serves a dual purpose of introducing the basics of the Tropos methodology as well as describing how we use it to specify a choreographed interaction. We give a brief introduction to the methodology, discuss its main modeling concepts, and show how we use them to represent a choreographed interaction. In particular, we introduce a diagrammatic notation for capturing the global view, local views, and their interrelations. We also detail how behavior of interaction participants is described using formal annotations of Tropos models. Other than a minor notational extension to express ordering constraints using dependency annotations, we do not claim a contribution to Tropos modeling notation or analysis activities in this chapter. Nevertheless, we propose a usage of Tropos models suitable for representing choreographed interactions, whereas typically Tropos has been used for modeling centrally-coordinated interactions (Desai et al., 2009).

4.1 Introduction to Tropos

Tropos originated as a methodology for building agent-oriented software systems (Bresciani et al., 2004). Tropos builds on the i* methodology (Yu, 1995), originally developed to support reasoning about early requirements, i.e. requirements at a high-level of abstraction and at an early stage of system development. Tropos carries over the benefits of the i* framework of covering early requirements analysis phases. Modeling early requirements helps deepen the understanding of the problem being solved and its context and rationalize the inter-dependencies between agents, software and human. One motivation behind i* was to develop a rich conceptual model for processes involving multiple participants (Yu, 1997). SOC can thus benefit from this conceptual model in specifying multi-participant choreographed interactions.

In addition to supporting early requirements analysis, Tropos covers late requirements analysis, architectural design, and implementation phases (Castro et al., 2002) thereby supporting reasoning across all phases of development. A typical top-down application of Tropos for system

analysis and design involves identifying stakeholders and representing them as roles, identifying objectives of each stakeholder and representing them as goals, outlining the context of the system to be developed, and relating it to roles via dependencies. Throughout the development process, stakeholder goals are successively refined into sub-goals and eventually operationalized by activities performed by each role.

Tropos provides a graphical modeling notation backed by a formal metamodel (Giunchiglia et al., 2002). It is also accompanied by formal reasoning techniques as well as automation tools (Giorgini et al., 2008). The methodology has been applied in various contexts including security modeling and analysis, goal-based risk analysis, and high-variability design. It has also been applied in the context of SOC, albeit for design of service implementations (Lau & Mylopoulos, 2004) and analysis of orchestrated service systems (Kazhamiakin et al., 2004).

4.2 Diagrammatic Specification of Interactions in Tropos

Of particular relevance to representing choreographed interactions are two types of Tropos diagrams: Role-Dependency (RD) diagrams and Goal-Activity (GA) diagrams. RD diagrams originated in the i* framework for reasoning about how a system to-be-developed is situated in its organizational environment. RD diagrams focus on intentional relationships between organizational roles and allows analysis of opportunities and vulnerabilities associated with these relationships. GA diagrams also originated in the i* framework for modeling stakeholder goals and rationalizing alternative means for achieving them. GA diagrams enable reasoning about how stakeholders goals are achieved by activities they perform and how they are impacted by their environment.

RD and GA diagrams provide a conceptual framework suitable for representing the global and local views of an interaction, respectively. Using the medical example introduced in chapter 2, we describe how we use RD diagrams for representing the global point of view of an interaction and GA diagrams for capturing the local view for each interaction participant. Additionally, we describe a hybrid diagram type that derives from the "Strategic Rationale" model (Yu & Mylopoulos, 1994) to combine the two points of view.

4.2.1 Global View of an Interaction in Tropos

Whereas choreography specifies messaging between a set of roles from a global point view, Tropos Role-Dependency (RD) diagrams capture interaction requirements from a global point of view. RD models capture interacting roles, goals motivating them to interact, and inter-dependencies driving the interaction between them. RD modeling activities involve identifying interacting roles, identifying goals associated with each role, and rationalizing their inter-dependencies for achieving their goals.

Figure 4.1 depicts an RD diagram for the medical example introduced in chapter 2. There are three roles in the example, each of which represents an abstract participant: Patient, Medical Provider (MP), and Doctor. Actual participants aim to achieve the goals associated with the role they play in an interaction. Goals associated with a role are attached to the circle representing the role. A goal is an objective which is achieved when a certain state of the world is reached or prevented. For instance, "Get Treatment" captures a Patient's goal, which is achieved when the Patient has received the desired treatment.

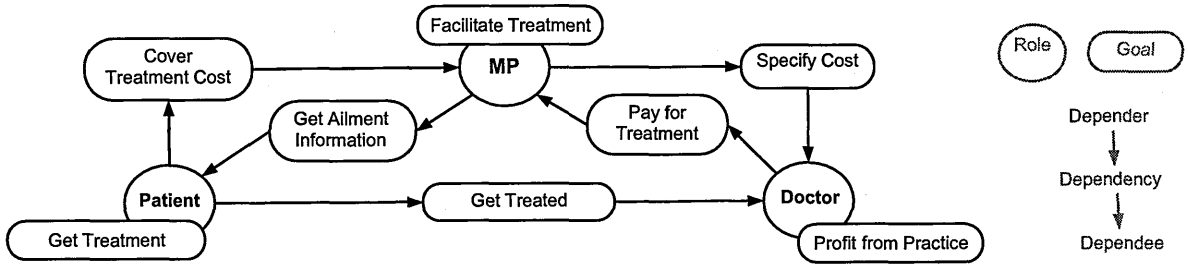


Figure 4.1 High-level Role-Dependency diagram for the medical interaction

Roles depend on each other for fulfilling their goals, hence the need to interact. A "goal dependency" represents delegation of goal-fulfillment responsibility from a depender role to a dependee role. For example, a Patient depends on the Doctor to "Get Treatment" and depends on the MP to "Cover Treatment Cost". In later phases goal dependencies between roles are refined into "activity dependencies" and "resource dependencies" (Bresciani et al., 2004) that capture their operational conditional obligations. A refinement of the RD diagram of Figure 4.1 is shown in Figure 4.2. A refined RD diagram depicts inter-role delegation of responsibility to perform

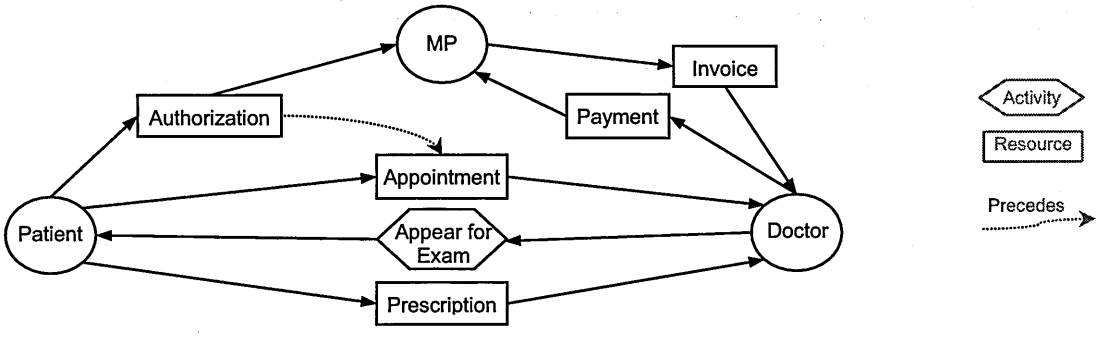


Figure 4.2 Refined Role-Dependency diagram for the medical interaction

activities and furnish resources. An activity is an abstraction of a course of action with well-defined pre- and post-conditions. For instance, the Doctor depends on the Patient to perform "Appear for Exam" activity. Resources are physical or informational entities. For example, the Doctor depends on the MP for providing a "Payment", which can be achieved either by physically mailing a check or electronically via wire transfer.

Although Tropos allows specification of constraints on the progression of an interaction from a local point of view, this feature is missing from a global point of view. As such, specification of relations between dependencies is lacking (Telang & Singh, 2009). To fill this gap we propose a notational extension for expressing precedence between dependencies. We propose using a "precedence link" to denote a constraint on fulfillment of dependencies imposed by the global context of an interaction. One such precedence constraint is specified in the diagram: a Patient is required to obtain an "Authorization" prior to attempting to obtain an "Appointment". Dependency precedence annotations constrain the progression of an interaction by constraining the order in which participants fulfill their obligations. The implication in the aforementioned example is that participants are obliged to ensure that a patient never obtains an appointment unless they have obtained an authorization first.

RD diagrams outline the context of an interaction; they specify interacting roles, goals they desire to achieve, and inter-dependencies that allow them to achieve their goals. The global view provided by RD diagrams abstracts away from internal details of *how* each role goes about performing activities to achieve their goals, which is only revealed in the local view for the role.

4.2.2 Local View of Interaction Participants in Tropos

To achieve goals associated with their role, each participant needs to perform activities in the course of an interaction. Goal-Activity (GA) diagrams capture the local point of view of a role in terms of activities to be performed, constraints that govern execution of these activities, as well as how they relate to achievement of goals. Through iterative refinement, high-level goals are refined into finer-grained goals and eventually into activities whose execution achieves the goals. Refinement serves to break down a "parent" goal into "sub-goals" and "sub-activities". The local view of a role is depicted inside an oval corresponding to that role. A participant playing the role is responsible for performing all activities and achieving all goals inside the oval. Figure 4.3 shows a GA diagram for the Doctor role.

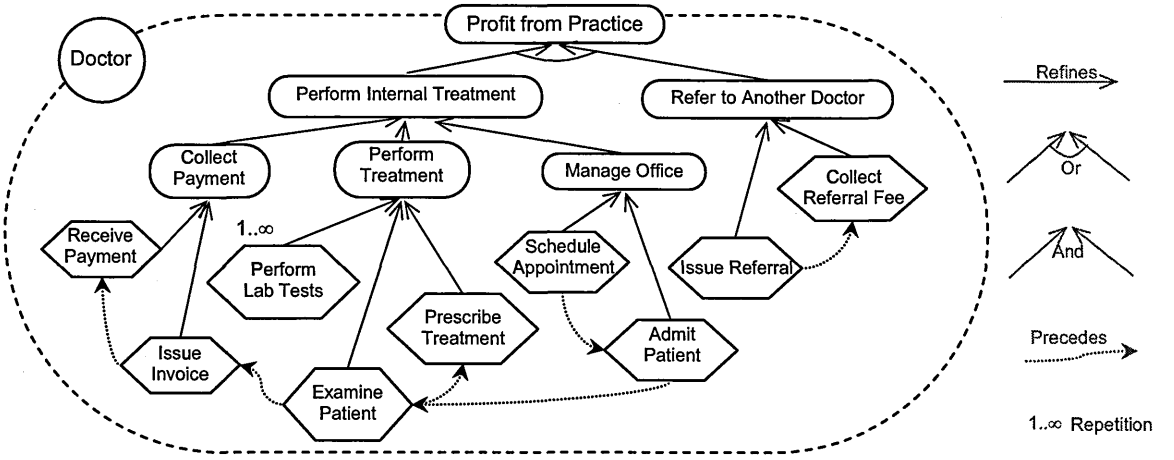


Figure 4.3 Goal-Activity diagram for the Doctor role

A Doctor has a choice between two alternatives to achieve "Profit from Practice" goal. This is represented by refining the goal into "Perform Internal Treatment" and "Refer to Another Doctor", which is an example of "OR" refinement. On the other hand, the Doctor decides to refine "Perform Internal Treatment" into three sub-goals "Perform Treatment", "Collect Payment", and "Manage Office". In this case, the achievement of the goal is contingent on achieving all three sub-goals, which is an example of an "AND" refinement.

Eventually, a fine-grained goal is refined into activities whose execution leads to fulfillment of the top-level goal. Business policies and data flow requirements may dictate a certain ordering

between activities, which is represented using "Precedence" links. Precedence links constrain the order of activity execution. For example, the Doctor requires a Patient to schedule an appointment before they appear for an exam which is represented as a precedence link between "Schedule Appointment" and "Admit Patient" activities.

An implicit 1-to-1 relation is assumed between each sub-activity or sub-goal and its parent unless repetition is explicitly specified. The diagram in Figure 4.3 implies that "Perform Treatment" activity may involve performing one or more "Perform Lab Test" activities. For example, a Doctor may order an X-ray at the beginning of treatment and another one at the end.

4.2.3 Combined Local-Global Model for Detailed Interaction Specification

Whereas RD diagrams depict inter-dependencies between participants, they do not specify the origin of inter-dependencies in participants' local models. On the other hand, GA models specify the local view of each role but they do not show its dependencies on other models. To fully specify an interaction, a combination of both diagrams is needed. This type of "Combined Local-Global" (CLG) diagram first appeared in Tropos literature in (Fuxman et al., 2003). Figure 4.4 shows the CLG diagram resulting from combining local models (with some details omitted for the sake of clarity) of the three interacting participants in the medical interaction with the global model of the interaction. Similar to messaging specification of an interaction, a CLG model represents a prototypical interaction or a template for which many instances may be instantiated.

A consistent set of local models and a global model are required to construct a valid CLG. That is, each dependency appearing in the global model must also appear in the CLG. Each dependency appearing in the RD diagram of Figure 4.2 links a depending activity in a local view of one role to a dependee activity in another. Attachment of dependency ends to activities in participant local models indicates their recognition of responsibility assignment. As such, a CLG captures an agreement between stakeholders on the specification of an interaction. The CLG diagram in Figure 4.4 captures the same medical interaction specified using WS-CDL in Figure 2.2, but at a higher level of abstraction that ties the interaction to business activities and goals. This level of abstraction does not specify the medium of execution of each activity; some

activities may be performed physically or electronically. For instance, Figure 4.4 still does not specify whether an MP mails a check or provides "Payment" electronically.

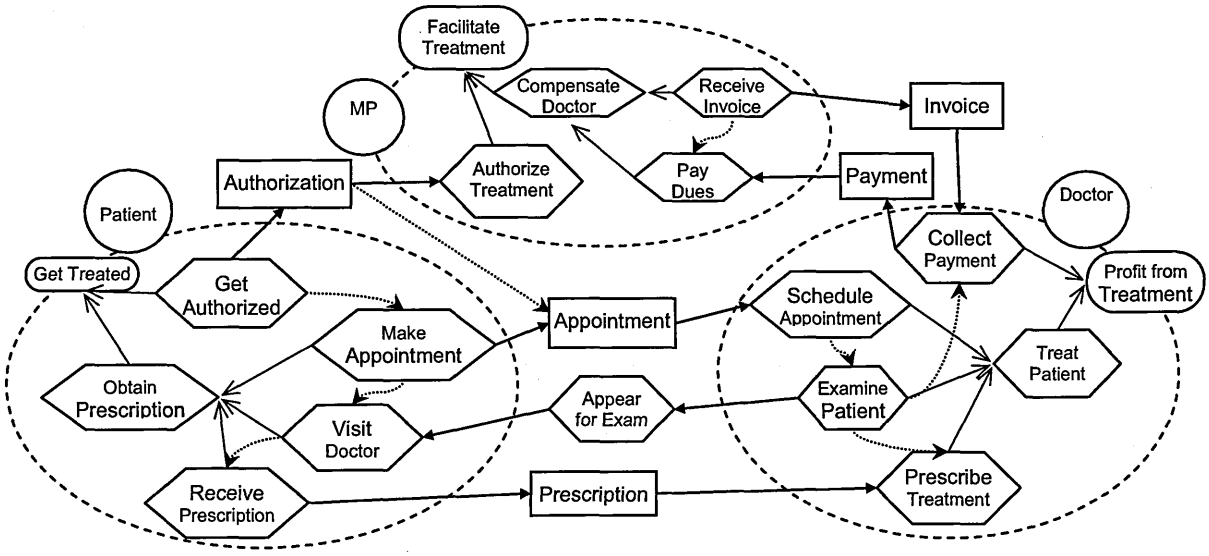


Figure 4.4 Combined Local-Global model of the medical interaction

4.3 Temporal Specification of Interactions in Formal Tropos

Formal Tropos (FT) (Fuxman et al., 2001) is an extension of Tropos that endows the diagrammatic specification with formal semantics. Using FT, elements of Tropos models are annotated with Linear-time Temporal Logic (LTL) formulas. FT annotations of Tropos model elements constrain participants' behavior and specify valid temporal progression of an interaction. FT supports several LTL operators for specifying behavior, those of which we use in this thesis are defined in Table 4.1. FT enables model checking (Clarke et al., 1999) for asserting properties of relatively large Tropos models (Fuxman, 2001).

Table 4.1 LTL Operators Applied to a Formula

Formula	Requirement Expressed By Formula
F *f*	Formula *f* is either true now or that it becomes **eventually** true in some future state
G *f*	Formula f should hold in the current state and **always** holds in all **future** states
O *f*	Formula *f* is either true now or that it was **once** true in some past state
H *f*	Formula *f* is true in the current state and was **always** true in all **past** states

4.3.1 Formal Tropos Classes and Instance Lifecycle

In FT, each Tropos model element is represented as an FT "class", of which many instances may be created during an "execution" of a model. FT classes and instances are analogous to classes and objects in object-oriented languages (Meyer, 1997). At any point in time, the state of an execution is equivalent to the collective state of all instances. Execution progresses as instances transition from one state to another.

Figure 4.5 shows an FT specification for the "Make Appointment" activity class and the "Appointment" dependency class, parts of which can be automatically obtained from Tropos diagrams by applying some heuristics (Fuxman et al., 2003). Each class has a list of attributes which hold the state of instances of that class as well associations with other instances. "Appointment" class has an attribute that specifies the type of "ailment" the patient suffers from and an attribute that references the associated instance of "MakeAppointment" class. The special attributes "depender" and "dependee" represent the two roles in a dependency class, while the attribute "role" associates a model element with the local view in which it appears, e.g. associate the "MakeAppointment" activity class with Patient. Finally, the special attributes "self" and "super" are used to refer to an instance of the class being specified and its parent instance, respectively. That is, "super" appearing in the specification of "Make Appointment" refers to its parent activity, "Obtain Prescription" as per figure Figure 4.4.

Activity MakeAppointment
Role Patient
Creation condition ¬**Fulfilled**(super)
Fulfillment condition
∃ a:Appointment
(a.depender = role
∧ a.makeAppointment = self ∧ **Fulfilled**(a))

Dependency Appointment
Depender Patient
Dependee Doctor
Attribute ailment: AilmentType
Attribute makeAppointment: MakeAppointment
Invariant Fulfilled(self) → G **Fulfilled**(self)
Creation condition ¬**Fulfilled**(makeAppointment)
Fulfillment condition
∃ sa:ScheduleAppointment
(sa.role = dependee ∧ **Fulfilled**(sa))

Figure 4.5 Formal Tropos annotations for an activity and a dependency

LTL formulas specify constraints on states and transitions of instances, thereby constraining model execution and specifying valid behavior of interacting roles. Whereas invariants specify conditions that do not vary with time, creation and fulfillment conditions specify when an instance is created (instantiated) and when it transitions to a "fulfilled" state. Creation and

fulfillment conditions specify the lifecycle of model elements as bracketed by two critical events: creation event and fulfillment event.

4.3.1.1 Creation Conditions and Creation Event

The creation event of a goal or of a dependency occurs at the moment at which a participant begins to desire the goal, or need the dependency to be fulfilled. Conditions under which a creation event may occur are called creation conditions. In Figure 4.5, an instance of "Appointment" dependency will be created if there is an instance of "MakeAppointment" activity that needs to be fulfilled. For an activity, the creation event occurs at the moment at which the participant is required to start performing it. We use $Cr(\alpha)$ to denote the creation condition of an instance α and use α_{cr} to denote the creation event of α.

4.3.1.2 Fulfillment Conditions and Fulfillment Event

Fulfillment events occur when a goal is achieved, an activity is completed, or a resource is made available. For a fulfillment event to occur the corresponding fulfillment conditions must hold. In Figure 4.5, an instance of "MakeAppointment" activity is fulfilled when the associated "Appointment" dependency has been fulfilled, i.e. when Patient has obtained an appointment, whereas an instance of "Appointment" is fulfilled when the Doctor has completed the activity of scheduling an appointment. We use $Fi(\alpha)$ to denote the fulfillment condition of an instance α and use α_{fi} to denote the fulfillment event of α.

4.3.1.3 Invariant Constraints

Invariant constraints of a class define conditions that should hold at any point in time for all instances of that class. In Figure 4.5, the "Appointment" dependency class declares an invariant specifying that once an instance of the dependency has been fulfilled it remains fulfilled, i.e. a Doctor is not allowed to cancel an appointment once it has been scheduled.

4.3.2 Ordering of Interaction Events

Whereas a message-oriented specification of an interaction is composed of message sending and receiving events, FT specifies progression of an interaction in terms of dependency/activity

lifecycle events. Temporal specification of a CLG in FT encompasses constraints on ordering of lifecycle events that occur during the interaction specified by the CLG, i.e. interaction events. To facilitate reasoning about a CLG, we explicitly capture these constraints using a binary precedence relation P over interaction events. We define P as follows: for two interaction events e_1 and e_2, $P(e_1, e_2)$ designates that e_2 must not occur before e_1 occurs in any valid instantiation of the interaction specified by the CLG. The relation P is transitive, i.e.

$$\mathbf{P(e_1, e_2) \wedge P(e_2, e_3) \rightarrow P(e_1, e_3)}$$

An implication of $P(e_1, e_2)$ is that if Φ_1 is a condition that triggers e_1 and Φ_2 is a condition that triggers e_2 then Φ_2 must not occur before Φ_1. That is:

$$\mathbf{P(e_1, e_2) \quad \leftrightarrow \quad \Phi_2 \rightarrow O\ \Phi_1}$$

The proof follows by contradiction: assume that Φ_2 held before Φ_1 held then e_2 would have occurred before e_1. In particular, for any Tropos model element X, since the creation event of X must occur before its fulfillment event, its fulfillment condition implies that its creation condition must have held at some point in the past. That is, $P(X_{cr}, X_{fi})$ is always true and so is the formula:

$$\mathbf{Fi(X) \rightarrow O\ Cr(X)}$$

We use an event precedence graph for visualizing pairs of the relation P. Figure 4.6 is an example of an event precedence graph depicting two pairs of the relation P, namely $P(e_1, e_2)$ and $P(e_2, e_3)$. Nodes in a precedence graph represent interaction events and edges represent precedence between them. A multi-edge path between two nodes in a graph represents transitive precedence between the corresponding events.

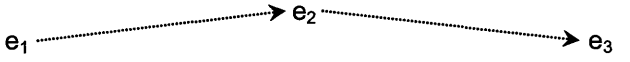


Figure 4.6 Event precedence graph for representing event precedence relation

4.4 Chapter Summary

This chapter demonstrated how we represent a choreographed interaction in Tropos using the medical interaction of Chapter 2 as an example. We introduced the Tropos methodology and its diagrammatic notation. We described how Role-Dependency (RD) diagrams capture the global view of an interaction in terms of roles and their dependencies. We described how the local view of each role is captured using Goal-Activity (GA) diagrams. We have also shown, by combining RD and GA diagrams, how to construct an overall specification of an interaction in the form of a CLG. A CLG can be annotated with Formal Tropos (FT) to specify temporal progression of an interaction by constraining interaction events. We defined a binary relation 'P' to facilitate capturing constraints on ordering of interaction events. Temporal annotations introduced in this chapter and their use in relating interaction events are crucial for the next 3 chapters.

Chapter 5. Relating Interaction Specification Viewpoints

Each participant in a choreographed interaction aims to achieve business goals relevant to their local viewpoint. Their local view embodies activities and business policies motivated by their goals. On the other hand, a neutral observer requires a global viewpoint that abstracts away from the particulars of each participant while enabling it to oversee an interaction. Specifying and adapting a choreographed interaction requires relating these disparate viewpoints.

Specifying a choreographed interaction also involves artifacts at different levels of abstraction. Requirements models of an interaction embody business goals behind an interaction. Architects use these models to reason that an interaction adequately addresses business goals of participants. On the other hand, an interaction protocol is inherently an operational representation intended for consumption by machines. To ensure that these two representations are consistent, we need to establish a relation between them.

This chapter reports on the first of our contributions: relating the artifacts involved in specifying a choreographed interaction. First, we propose four viewpoints that aim to separate concerns of stakeholders in an interaction. We argue that Tropos dependencies play a central role in establishing relations between the viewpoints, and hence we analyze characteristics of dependencies. We employ this analysis to relate local and global requirements as well as relate global requirements to choreographed messaging.

5.1 Separating Stakeholder Concerns

Two types of stakeholders are concerned with the specification of a choreographed interaction: the interacting participants and a global observer. Each participant is a stakeholder interested in fulfilling business needs relevant from their point of view, while the global observer is a stakeholder interested in facilitating an interaction from a neutral point of view. On the one hand, each stakeholder has business-level concerns regarding the achievement of goals, seizing business opportunities, and mitigation of risks. On the other hand, each stakeholder has

operational concerns regarding coordination of activities and compliance with messaging obligations. This separation of stakeholders and concerns results in four interaction specification viewpoints.

5.1.1 Four Interaction Specification Viewpoints

Figure 5.1 depicts separation of concerns according to type of stakeholder and type of concern into four viewpoints, each represented by a quadrant. Each viewpoint encapsulates a set of concerns from the point of view of a stakeholder type as follows.

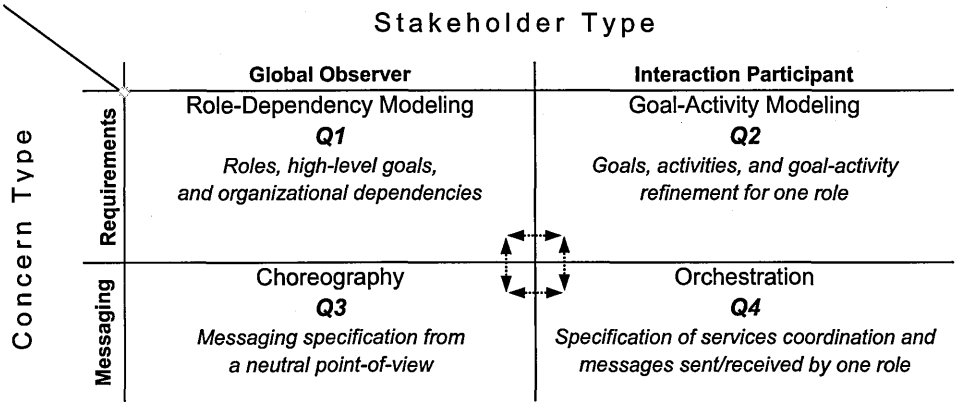


Figure 5.1 Four interaction specification viewpoints.

5.1.1.1 Q1: Requirements from a Global Point of View

The global observer in an interaction is typically a regulatory agency aiming to facilitate the interaction. The regulatory agency acts as a neutral stakeholder whose objectives are global, i.e. not specific to any of the participants, but rather broadly benefits all potential participants. For instance, the global objective could be promoting trade (Baglietto et al., 2002) or enabling advancement across an industry sector. To achieve such objectives, a regulatory agency needs to ensure viability of an interaction and encourage participants to join. This calls for means to rationalize responsibilities of interacting roles, to ensure fairness of allocation, and to help participants mitigate risks entailed in delegation of responsibilities. RD diagrams are a suitable tool for supporting these activities (Yu, 1997) as they capture the interacting roles, their high-level goals, delegation of goal-fulfillment responsibilities via dependencies, and risks that come with these dependencies:

- Rationalize goal-fulfillment responsibility: A Patient's expectation that an MP will "Cover Treatment Cost" is consistent with a Doctor's reliance on the MP for "Pay for Treatment" (Figure 4.1).

- Ensure fairness of responsibility allocation: Requiring a Patient to obtain treatment authorization from an MP is justified as the latter is responsible for covering the cost as specified in Figure 4.2.

- Mitigate risks involved in delegation: Although it is reasonable to assume that a Doctor has the necessary expertise to fulfill the "Specify Treatment Cost" goal (Figure 4.1), it may entail the risk that an MP gets over-charged by the Doctor. Identifying such risks drives further analysis to explore alternatives for mitigating them.

5.1.1.2 Q2: Requirements from a Local Point of View

The main concern of each participant is to ensure the achievement of goals motivating them to join an interaction. A participant needs to identify, represent, and analyze their business goals in order to share knowledge and better-understand business problems (Yu & Mylopoulos, 1994). Having decomposed their goals into more manageable finer-grained goals, a participant needs to determine how to go about achieving them. This calls for means to explore solutions for achieving goals and rationalize decisions made in choosing a solution. Specifying a solution involves identifying business activities, electronic and physical, whose execution leads to goal fulfillment, as well as detailing constraints that govern execution of these activities.

GA diagrams and their associated modeling techniques are suitable for addressing these concerns. A GA diagram depicts successive refinement of high-level goals into finer-grained goals and eventually into the activities assigned to one role. Through refinement, relations are established between goals and activities thereby enabling reasoning about how activities contribute to goal achievement.

5.1.1.3 Q3: Messaging from a Global Point of View

To facilitate execution of an interaction, the global observer needs to ensure that participants are able to interoperate, i.e. their expectations of each other are met. Ensuring interoperability requires accurate description of obligations that a participant will be committing to when joining an interaction. It is software agents, i.e. services and clients, which carry out the electronic part of an interaction by exchanging messages on behalf of participants. Thus, messaging obligations need to be specified as a machine-readable protocol that describes valid messaging sequences. Since the global observer is only concerned with the observable (and not internal) behavior of participants, a messaging protocol is specified from a global point of view. Furthermore, to enable interaction between heterogeneous platforms these obligations need to be described using standard platform-independent languages. Choreography description languages, such as WS-CDL, are adequate for this purpose; specifying standards-based, platform-independent messaging protocols from a neutral point of view.

5.1.1.4 Q4: Messaging from a Local Point of View

An interaction participant is likely to take part in many different types of interactions at the same time. For example, in addition to participating in the example medical interaction, a Doctor may participate in another interaction for conducting lab tests, a third for reporting their profits, and so on. Although from a global point of view these interactions can be treated separately, from a local point of view they overlap. A participant is thus interested in coordinating all their messaging activities. This serves the dual purpose of ensuring that execution of their activities complies with their internal business policies as well as with their external obligations towards all interactions. These concerns are addressed by orchestration languages, such as BPEL, which are used to specify messaging flows from a local point of view.

5.1.2 Consistency between the Viewpoints

Having identified interaction specification viewpoints and proposed representations for each of them, we need means for ensuring consistency between these representations. Given that there are two sets of artifacts (requirements and messaging) and two types of stakeholder viewpoints

(global and local) we need to establish four types of relations. This translates to relating pairs of adjacent quadrants in Figure 5.1:

1. (Q1-Q2) Consistency between local and global requirements is necessary for agreement between stakeholders on responsibility allocation and on an overall specification of an interaction. Relating the local and global viewpoints of interaction requirements enables change propagation between viewpoints, thereby supporting collaborative adaptation. Dependencies in a CLG tie together local models and the global model; we examine this relation in depth in section 5.3.

2. (Q1-Q3) Consistency between global requirements and choreographed messaging is necessary for ensuring that messaging specification satisfies the requirements. We establish a relation between dependencies in a global model to units of messaging that realize these dependencies, as we'll detail in section 5.4. By combining this relation with that between local and global requirements (Q1-Q2) we enable automatic derivation of choreographed messaging from a CLG as detailed in chapter 6.

3. (Q2-Q4) Consistency between a participant's local requirements and orchestrated messaging enables the generation of a messaging specification that satisfies the requirements, or verifying that a given messaging specification satisfies the requirements. As has been detailed in section 3.1.2.1, Kazhamiakin et al. (2004) have proposed a framework through which the lifecycle events of a participant's business activities are related to orchestrated messaging that realizes these activities. As this issue has been addressed, we will not cover it further.

4. (Q3-Q4) Consistency between orchestrated and choreographed messaging enables verifying compliance of internal processes of participants with a choreographed interaction protocol. As detailed in section 3.2.3, Foster (2006) proposed a framework for automated consistency checking between a set of BPEL processes with a WS-CDL protocol. As this issue has been addressed, we will not cover it further.

Due to the central role that dependencies play in establishing the Q1-Q2 relation as well as the Q1-Q3 relation, we devote the next section to examining dependency characteristics.

5.2 Analysis of Dependency Characteristics

To understand how dependencies play a role in relating interaction specification views Q1-Q2 and Q1-Q3, we examine their control and data flow implications. Additionally, to relate dependency lifecycle events to messaging events we analyze characteristics of dependency fulfillment. We also present a notation for capturing these characteristics.

5.2.1 Dependency Data and Control Flow Characteristics

Dependencies in a CLG represent delegation of responsibility between activities of interacting participants. At runtime, data and control flow between activities realize the delegation and fulfillment of responsibility. Control flow between participants governs the progression of an interaction whereas data flow implies communication between them, e.g. via messaging. Understanding these flows is essential for both relating participants' views as well as relating dependency lifecycle events to messaging events. The business meaning of a dependency determines the type of flow it implies as exemplified by dependencies in Figure 4.2:

- "Appear for Exam" designates control flow. To fulfill the dependency a Patient has to perform "Visit Doctor" activity by showing up at a Doctor's office at which point she transfers control of interaction progression to the Doctor, where the Doctor can start performing "Examine Patient" activity.

- "Prescription" designates unidirectional flow of data from Doctor to Patient. To fulfill the dependency a Doctor is required to provide specification of medications to a Patient. From the Patient's point of view the dependency is fulfilled when they have received the specification of medications.

- "Authorization" designates both data and control flow. To fulfill the dependency an MP is required to provide a treatment authorization, which includes data such as an

authorization number, in doing so the MP also relinquishes control to a Patient who may then proceed to request a Doctor appointment.

- "Appointment" designates bidirectional data flow. The two flows are realized in Figure 2.1 and Figure 2.2 via the "Appointment Request" and "Appointment Confirmation" message. To fulfill the dependency, a Doctor has to confirm an appointment corresponding to an appointment slot that a Patient had requested.

To summarize, a dependency implies data or/and control flow between participants. Where a dependency implies unidirectional flow, the flow is in the opposite direction of dependency arrows, i.e. from the dependee to the depender, e.g. a Doctor providing "Prescription" to a Patient. A unidirectional dependency is a special case of a dependency where no data flow from the depender to the dependee is implied. A dependency is "bidirectional" if it implies a two-way flow, one from depender to dependee followed by another in the reverse direction, e.g. an "Appointment Request" followed by an "Appointment Confirmation". For a bidirectional dependency, we refer to flow from depender as "request" and flow from dependee as "response". For a unidirectional dependency, we use the same term, "response", to refer to flow from dependee for consistency, even though there is no corresponding request. Where a request corresponding to a (bidirectional) dependency D is realized via sending a message, "D-request", we denote the "request sent" event by D_{rs}, and where the response is realized via receiving a message, "D-response", we denote the "response received" event by D_{rr}.

The business context and participant requirements dictate whether a dependency is unidirectional or bidirectional. For example, a Patient is required to specify a requested appointment slot, by sending an "Appointment Request" message, in order to get a response from a Doctor, and hence the bidirectional flow of the "Appointment" dependency. On the other hand, a Patient is not required to request a prescription as it is provided by the Doctor after having examined the Patient, and hence the unidirectional flow of the "Prescription" dependency. Bidirectional dependencies allow an architect to construct more modular CLGs, where logically related data flows can be grouped. For instance, instead of representing an invoice and the corresponding payment separately using two unidirectional dependencies

("Invoice" and "Payment" in Figure 4.4) an architect may choose to combine the two flows into a single bidirectional "Payment" dependency and dispense with the two unidirectional dependencies, which we do hereafter.

5.2.2 Dependency Fulfillment Characteristics

We examine the properties of dependency fulfillment which are essential for reasoning about the relation between fulfillment events and messaging events. Each dependency in a Tropos model is of a certain granularity and repetition. Also, each dependency can be associated with a medium of fulfillment as well as pre-conditions on its fulfillment.

5.2.2.1 Fulfillment Granularity

Dependencies denote delegation of responsibility at different levels of abstraction. As shown in the refined RD diagram of Figure 4.2, the goal dependencies of Figure 4.1 were refined into elementary activity and resource dependencies. In general, in a global view, a goal dependency is refined into activity/resource dependencies which in turn maybe refined into finer-grained elementary activity/resource dependencies and a set of constraints on their fulfillment. Thus we distinguish between coarse-grained and elementary dependencies, where the former are fulfilled only if their elementary sub-dependencies are fulfilled.

5.2.2.2 Fulfillment Repetition

Fulfillment of non-repeating dependency requires exactly one instantiation of the corresponding dependency class in any instance of an interaction. On the other hand, multiple instantiations may be required for a dependency marked as "repeating" to be eventually fulfilled. For instance, it can be specified that multiple instantiations of an "Appointment" dependency may be necessary. That is, a Doctor may refuse an appointment slot requested by a Patient causing the Patient to request a different one, and so on until they agree upon a slot.

5.2.2.3 Fulfillment Phenomenon

A dependency is fulfilled when its fulfillment condition becomes true. The depender in a dependency detects this state transition by observing a designated phenomenon (Jackson, 1996). Depending on the medium of the phenomenon, a dependency is classified as either "physical" or

"informational". Fulfillment of a physical dependency is associated with a physical phenomenon. For example, a patient arriving at the Doctor's office for examination is a physical occurrence that indicates fulfillment of "Appear for Exam" dependency. On the other hand, fulfillment of an informational dependency is contingent on the depender receiving some required "fulfillment information". In an electronic interaction, information becomes available to a depender when they receive a message (sent by the dependee). For instance, the "Authorization" dependency is fulfilled when a patient has received a message indicating that treatment has been authorized.

5.2.2.4 Fulfillment Precondition

The responsibility of fulfilling a dependency is conditional. A dependee is held responsible for fulfilling a dependency only when a certain dependency-specific precondition holds. Participants are autonomous and may deviate from the specified interaction protocol. Such a deviation may occur if a depender causes a dependency to be instantiated when the precondition does not hold. When the dependee learns about the instantiation she has two choices:

- Wait until the precondition becomes true then fulfill the dependency.
- Immediately "dismiss" the dependency indicating that it will never be fulfilled.

For example, assume a Patient is required to pay a monthly fee to their MP in return for covering treatment cost. Additionally, assume that an MP requires as a pre-condition to fulfilling an "Authorization" dependency that a Patient has already paid their fee for the current month. If an MP is requested to provide authorization for a Patient who has not paid their fees, an MP may either wait until the Patient has paid their dues and then authorize treatment or immediately decline the request for authorization, i.e. dismiss the dependency. To represent the latter case, we define the notion of dependency "dismissibility", where a dependency D may be marked as "dismissible" under a dismissibility condition Di(D). The semantics is that that the dependee will never fulfill D if the condition Di(D) holds. In temporal logic, this is represented by including the following invariant in the specification of the dependency FT class:

$$\mathbf{Di(D) \rightarrow G \neg Fi(D)}$$

Thus, in addition to the "fulfilled" state, a dismissible dependency has another terminal state, the "dismissed" state (Figure 5.2).

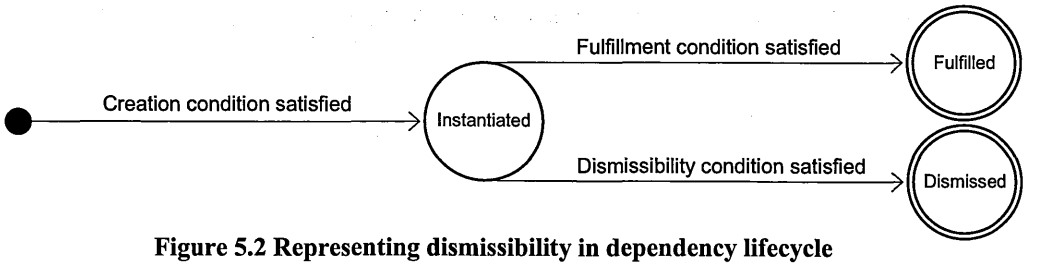


Figure 5.2 Representing dismissibility in dependency lifecycle

As a corollary of dismissibility, the creation condition of a dismissible dependency should ensure that a dependency is not immediately dismissed after being instantiated, or otherwise the dependency instance is useless. That is, the creation condition of a dismissible dependency should imply that the dismissibility condition is not satisfied:

$$\mathbf{Cr(D) \rightarrow \neg Di(D)}$$

For instance, the creation of the "Authorization" dependency should imply that the Patient has already paid their fees. Otherwise, if an instance of "Authorization" is created at a point in time where the Patient had not paid their fees it will be immediately dismissed.

5.2.3 Notation for Capturing Dependency Characteristics

To specify properties of a dependency D we define the following notation:

Notation	Denotes
BD(D)	Predicate whose value is true iff D is bidirectional (section 5.2.1)
BD D	Bi-directional dependency in a CLG.
Many(D)	Predicate whose value is true iff D is repeating (section 5.2.2.2)
D P	Physical dependency in a CLG
Di(D)	Dismissibility condition of D (section 5.2.2.4)

As discussed in section 5.2.1, a depender observes messaging events associated with messages that realize a dependency D which are denoted as follows:

Notation	Denotes
D-Request	Request message associated with a bidirectional dependency D.
D-Response	Response message associated with a dependency D.
$\mathbf{D_{rs}}$	Messaging event associated with sending "D-Request".
$\mathbf{D_{rr}}$	Messaging event associated with receiving "D-Response".

5.3 Relating the Global and Local Views of Interaction Requirements

A dependency arises when an activity in a depender's local view delegates responsibility to another participant, i.e. the dependee. On the dependee side, an activity is assigned (by the dependee) the responsibility of fulfilling the dependency. Thus, a global view is consistent with a set of local views only if every delegation of responsibility in a local view is represented as a dependency that ties two activities: a depender activity and a dependee activity. This relation is manifested in a CLG diagram (Figure 4.4). Furthermore, any dependency precedence constraints must be consistent with constraints specified in local views. In Figure 4.4 precedence between "Get Authorized" and "Make Appointment" activities is consistent with precedence between "Authorization" and "Appointment" dependencies.

By relating each local view with the global view we also establish a relation between local views. This relation is manifested in the FT of Figure 4.5 where "Make Appointment" activity is not fulfilled until the "Appointment" dependency has been fulfilled, which is in turn only fulfilled when the "Schedule Appointment" activity has been fulfilled. Thus, fulfillment of "Schedule Appointment" is necessary for the fulfillment of "Make Appointment", which ties together the states of the two views. In general, for any dependency D whose depender is an activity α and dependee is an activity β, α cannot be executed to completion until β has completed and made information required to fulfill D available. Additionally, the dual flow applies only for a bidirectional dependency, that is for a BD(D), β cannot execute to completion

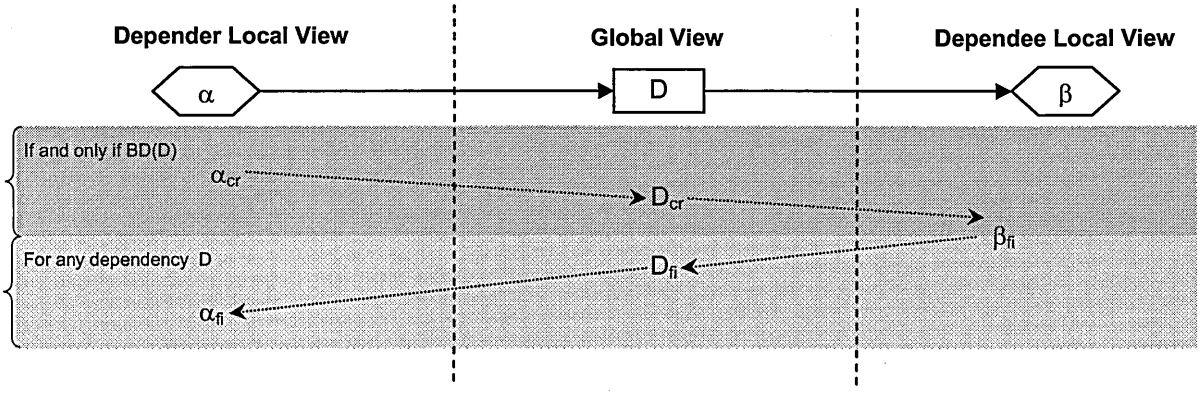


Figure 5.3 Graph relating dependency lifecycle events to those of depender and dependee activities

until α has supplied information, e.g. requested appointment slot, required for D to perform its work. These relations between dependency and activity lifecycle events tie the global and local views as summarized in Figure 5.3.

5.4 Relating Requirements to Choreographed Messaging

Dependencies imply data and control flow between participants. In a message-oriented interaction, these flows are realized via message exchanges. To structure our argument for relating dependencies and messaging, we construct a classification of dependencies utilizing the analysis in section 5.2. The classification allows us to formalize the temporal relation of dependency lifecycle events to messaging events.

5.4.1 Messaging-Oriented Dependency Classification

Figure 5.4, depicts a classification of dependencies with respect to their relation to messaging events. To narrow down the scope of the discussion on relating lifecycle events of a single dependency to messaging events, we consider the following aspects of a dependency:

- **Granularity**: Coarse-grained dependencies in a global model are refined into elementary dependencies in a CLG. Thus for the purpose of relating CLG dependencies to messaging we only need to consider elementary dependencies. Coarser-grained dependencies are indirectly related to messaging through refinement.

- **Phenomenon**: By definition, physical dependencies are fulfilled via means other than electronic messaging. Thus, for the purpose of relating lifecycle events of a ***single*** dependency to messaging events we only need to consider informational dependencies. However, physical dependencies may affect the overall sequence of messaging, which we consider in chapter 6.

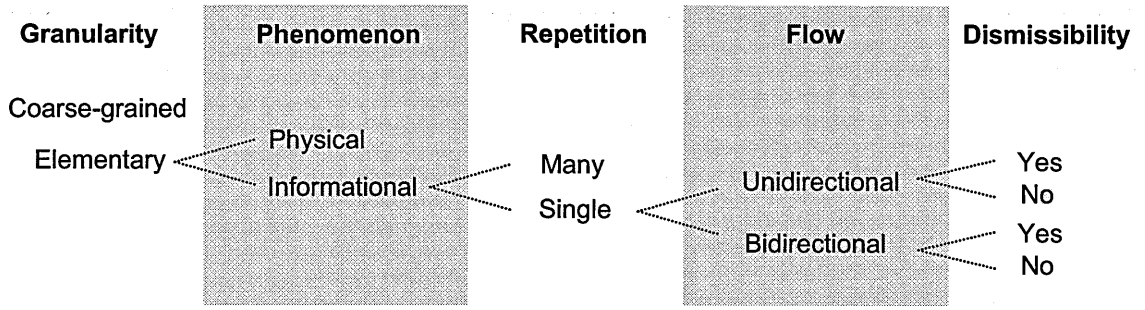


Figure 5.4 Classification of dependencies with respect to messaging

- **Repetition**: Several instances may be created of a repeating dependency D in a single instance of an interaction. Aside from data values, e.g. appointment time, all instances of D are structurally and behaviorally identical. Thus, to relate messaging events to a dependency instance we only need to consider a single representative instance of D.

Therefore, for establishing a relation between lifecycle events of a dependency instance and messaging events we only need to consider elementary informational dependencies, each of which can be either unidirectional or bidirectional and may or may not be dismissible.

5.4.2 Relation between Dependency Lifecycle Events and Messaging

A dependency may undergo three types of state transitions (Figure 5.2). It is the depender who observes the fulfillment or dismissal of a dependency, and thus the lifecycle of a dependency terminates at the depender's end. Let us consider the implications of each transition for messaging from the point of view of the ***depender***. Having excluded coarse-grained and physical dependencies, the following statements apply only to instances of elementary informational dependencies.

Fulfillment: To indicate that they have met a responsibility delegated to them via a dependency, a dependee has to communicate dependency fulfillment to the depender, via a message, after they have fulfilled that responsibility. Once the depender receives the designated message they determine that the dependency has been fulfilled. Therefore:

Dependency fulfillment is detected when a designated message is received from the dependee.

Creation: Whereas fulfillment of elementary informational dependencies is always associated with a message, their creation may or may not be associated with one. A unidirectional dependency does not require flow of data and/or control from the depender to the dependee (e.g. "Prescription"), but a bidirectional dependency does (e.g. "Appointment"). This flow needs to be realized at a point in the lifecycle of a dependency no earlier than creation and no later than fulfillment. Therefore:

Creation of a bidirectional dependency is followed by sending a message to the dependee.

Dismissal: For a dismissible dependency, when a dependee decides to dismiss the dependency she needs to communicate the dismissal to the depender via a message designated to indicate dismissal. The depender learns about dismissal when they receive the designated message. Therefore:

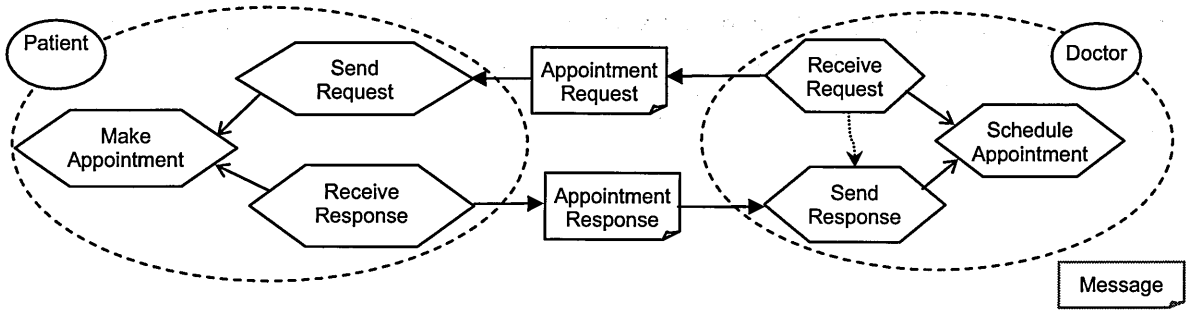
Dependency Dismissal is detected when a designated message is received from the dependee.

5.4.3 Messaging and Dependency Lifecycle Events in Formal Tropos

From the preceding discussion we conclude that an instance of a bidirectional elementary informational dependency is realized by two messages:

- A request message sent by the depender after the dependency has been instantiated.
- A response message received by the depender which fulfills the dependency.

To demonstrate the correctness of this conclusion we apply it to "Appointment" dependency. Figure 5.5 depicts the refinement of the "Appointment" dependency into two messages, "Appointment Request" and "Appointment Response", where each message is represented as a resource dependency. Messages exchanged between participants are specified as message-resource dependencies between "messaging activities" whose execution results in sending or receiving a message.



Activity MakeAppointment **Role** Patient **Fulfillment condition** ∃ sr: SendRequest (sr.super = self ∧ **Fulfilled**(sr)) ∧ ∃ rr: ReceiveResponse (rr.super = self ∧ **Fulfilled**(rr))	**Task** SendRequest **Role** Patient **Super** MakeAppointment **Creation condition** ¬**Fulfilled**(super) **Fulfillment condition** ∃ ar:AppointmentRequest (**Sent**(ar))	**Activity** ReceiveResponse **Role** Patient **Super** MakeAppointment **Fulfillment condition** ∃ ac:AppointmentResponse (**Received**(ar))

Figure 5.5 Refinement of activitis into messaging activities and dependency into messages

A message-receiving activity depends on a message-sending activity for providing a message over a communication channel. The corresponding FT specification of activities at the depender's end uses the predicates *Sent()* and *Received()* to assert that a message has been sent or received, respectively, by the depender (Kazhamiakin et al., 2004). Activities, "Make Appointment" and "Schedule Appointment", at the ends of "Appointment" dependency in Figure 4.4 have been refined into the messaging activities in Figure 5.5 as follows: In order to complete "Make Appointment" a Patient performs "Send Request" to send an "Appointment Request" message then later performs "Receive Response" to receive an "Appointment Response" message. Similarly, for every appointment, a Doctor schedules she performs a "Receive Request" followed by "Send Response".

In order to demonstrate that our proposal in the previous section of how to realize dependencies via messaging is consistent with Tropos model semantics, we construct the event precedence graph in Figure 5.6, which composes three sets of relations:

1. Relations between the depender's business activity and its child messaging activities as specified in Figure 5.5 (shown inside light grey shade, where α is "Make Appointment", αs is "Send Request", and αr is "Receive Response").

2. Relation between depender activity lifecycle events and dependency lifecycle events as depicted in Figure 5.3 (shown inside two-part dark grey shade).

3. Relation between dependencies and messaging that realizes these dependencies, at the depender's end, as we proposed in the previous section (shown inside medium grey shade). Recall that D_{rs} denotes message sending event, occurring after instantiation of D, and D_{rr} denotes message receiving event, leading to fulfillment of D. Both events are observed by the depender in D.

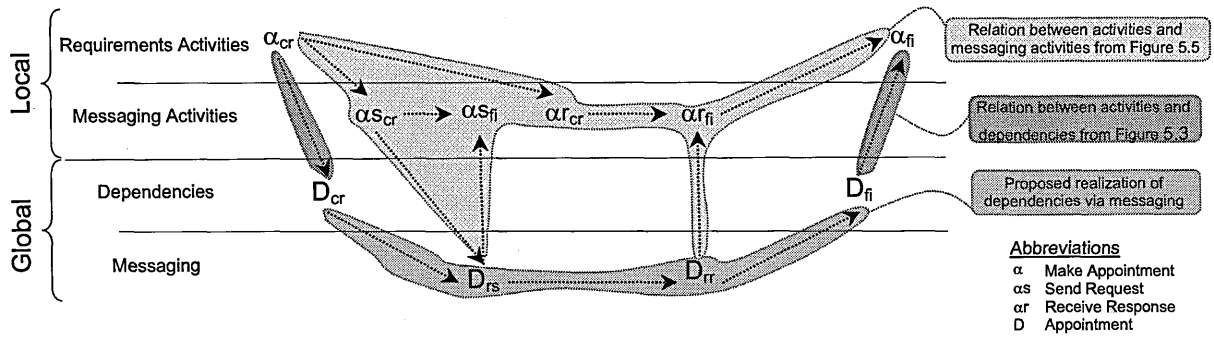


Figure 5.6 Event graph relating dependency lifecycle events to messaging events

As evident from the absence of cycles in the figure, the three sets are consistent. That is, our proposed realization of a bidirectional dependency as a pair of messaging events is consistent with the intrinsic semantics of a Tropos model. The correctness of the two precedence pairs that we suggest, that is $P(D_{cr}, D_{rs})$ and $P(D_{rr}, D_{fi})$, is demonstrated by these two observations drawn from the figure:

- Correctness of precedence between dependency creation event and sending of a request message, i.e. $P(D_{cr}, D_{rs})$: Creation of α transitively precedes request-message sending events (via the creation event of the message sending activity αs). That is, the creation of "Make Appointment" precedes the creation of its child "Send Request", where the creation of the latter triggers sending a request message. Our proposal for adding precedence from dependency creation to request-message sending event agrees with this already existing transitive precedence (via the creation event of the dependency). That is:

$$\mathbf{P(\alpha_{cr}, \alpha s_{cr}) \wedge P(\alpha s_{cr}, D_{rs}) \rightarrow P(\alpha_{cr}, D_{rs})}$$

is consistent with

$$\mathbf{P(\alpha_{cr}, D_{cr}) \wedge P(D_{cr}, D_{rs}) \rightarrow P(\alpha_{cr}, D_{rs})}$$

- Correctness of precedence between receiving a response message and dependency fulfillment event, i.e. $P(D_{rr}, D_{fi})$: Response-message receiving event transitively precedes the fulfillment event of α (via the message receiving activity αr). That is, receiving an "Appointment Response" message triggers the fulfillment of the "Receive Response" activity, which in turn precedes the fulfillment of its parent, "Make Appointment". Our proposal for adding precedence from the response-message receiving event to the dependency fulfillment event agrees with this already existing transitive precedence (via the fulfillment event of the dependency). That is:

$$\mathbf{P(D_{rr}, \alpha r_{fi}) \wedge P(\alpha r_{fi}, \alpha_{fi}) \rightarrow P(D_{rr}, \alpha_{fi})}$$

is consistent with

$$\mathbf{P(D_{rr}, D_{fi}) \wedge P(D_{fi}, \alpha_{fi}) \rightarrow P(D_{rr}, \alpha_{fi})}$$

Thus, our proposed realization of a bidirectional dependency via a pair of messages, a request message sent by the depender upon creation of the dependency and a response message received by the depender that triggers its fulfillment, is consistent with the inherent semantics of Tropos models. This conclusion is pivotal to establishing the relation between requirements and choreographed messaging, i.e. Q1-Q3 (Figure 5.1), which is the subject of the next chapter.

5.5 Chapter Summary

This chapter elaborated on the first of our contributions: relating the viewpoints involved in specifying a choreographed interaction. Two types of stakeholders were identified and their concerns with respect both to interaction requirements and messaging specification were discussed. This separation of concerns resulted in four views of interaction specification, namely global requirements, local requirements, global messaging, and local messaging. To provide a means for keeping the views consistent, we formulated relations between them. First, we detailed how we relate the global requirements of an interaction to the local requirements of each participant. This relation enables collaborative reasoning about adaptations to an interaction. Second, we presented a classification of Tropos dependencies and argued how they are realized in terms of messaging. In the next chapter, we combine the relation between dependencies and messaging with constraints specified in local views to automate the derivation of a messaging protocol from requirements models.

Chapter 6. From Requirements to Choreography Specification

This chapter reports on our second contribution: a systematic technique for deriving a messaging protocol from requirements models. To facilitate the derivation, we analyze constraints implied by links and annotations of Tropos models. We devise a transform that operates on Tropos models to generate a messaging protocol consistent with these constraints. We specify messaging protocols using an Abstract Choreography Description Language (ACDL), which we introduce next. To ground our approach in public standards, we define a mapping from ACDL to WS-CDL.

6.1 Abstract Choreography Description Language (ACDL)

Per the review in chapter 2, WS-CDL is the leading CDL. However, to avoid distractions introduced by the verbose XML syntax of WS-CDL, we use an Abstract Choreography Description Language (ACDL) to specify a messaging protocol. ACDL also serves the purpose of an intermediary language to avoid direct dependence of our approach on any particular CDL. Nevertheless, the semantics of ACDL constructs are consistent with WS-CDL, which makes translating an ACDL specification to a skeletal WS-CDL specification straightforward. Similar to WS-CDL, ACDL specifies choreographed activities and control flow between them. We use the term "execution" to refer to progression of the interaction corresponding to an ACDL specification in the same sense as choreography "life-line" (Kavantzas et al., 2005). The ACDL grammar specifies nine types of activities (Figure 6.1), whose semantics are defined as follows:

1. *Message*: Specifies message sending from one role to another along with a literal describing the message. Messaging is synchronous, that is a Message activity is only completed when the message has been received.

2. *Sequence*: Specifies sequential composition of activities, where an activity within a sequence may not start unless the preceding activity has completed. A "Sequence" activity is completed when the last activity in the sequence is completed.

3. *Parallel*: Specifies parallel composition of activities where individual branches within a "Parallel" may proceed concurrently. A "Parallel" activity is only completed when all branches are completed.

4. *Repetition*: Specifies conditional repetition where an activity is executed any number of times as long as a Boolean condition holds.

5. *Conditional*: Specifies conditional branching where exactly one of two activities is executed depending on the value of a Boolean condition.

6. *Choice*: Specifies choice between mutually exclusive activities at the discretion of one of the roles. A Choice completes when exactly one of the enclosing activities completes.

7. *Assignment*: Specifies assignment of a value to a Boolean variable. No explicit variable declaration is required, a variable is declared on first use.

8. *No Operation*: Specifies an activity that does not do any work.

9. *Failure:* Designates a failure of an interaction to complete and states a reason.

Choreography → Activity
Activity → Message | Sequence | Parallel | Repetition
| Conditional | Choice | Assignment | Failure | No Operation
Message → R_1 **Send** *MessageName* **To** R_2
Sequence → **Sequence {** Activity * **}**
Parallel → **Parallel {** Activity * **}**
Repetition → **While** *Condition* Activity
Conditional → **If** *Condition* Activity **Else** Activity
Choice → **Choice {** Activity * **}**
Assignment → *VariableName* = *Value*
No Operation → **Noop**
Failure → **Fail** *Reason*

Figure 6.1 Grammar of Abstract Choreography Description Language

6.2 Ordering Semantics of Tropos Models

The semantics of a CLG, implicitly and explicitly, impose constraints on ordering of interaction events. A few publications have made observations about constraints embedded in a Tropos model (Fuxman et al., 2004; Koliadis et al., 2006a; Mallya & Singh, 2006). However, none of them provided a complete formalization of these constraints and how to extract them from a Tropos model, a task that we undertake in this section. We extract ordering constraints associated with all constructs of a CLG, namely: dependency, refinement, precedence, and repetition. We express these constraints precisely in linear temporal logic and, where applicable, state their implications on the binary relation P over interaction events (see section 4.3.2), and illustrate them using the medical example. These constraints are central to deriving a messaging protocol from a Tropos model.

6.2.1 Dependency

In reference to section 5.3 and Figure 5.3, an instance of the "Make Appointment" activity is not fulfilled unless the associated "Appointment" dependency has been fulfilled, which in turn is not fulfilled until a Doctor has completed the associated "Schedule Appointment". In general, an activity α is not fulfilled until any dependency in which it is the depender has been fulfilled, and in turn the dependency is not fulfilled until the dependee activity, β, has been fulfilled. That is, fulfillment of β must precede fulfillment of D which in turn must precede fulfillment of α:

$$\forall D\ (D.depender = \alpha \wedge D.dependee = \beta) \rightarrow (Fi(\alpha) \rightarrow O\ Fi(D) \wedge Fi(D) \rightarrow O\ Fi(\beta))$$

i.e.

$$P(\beta_{fi}, D_{fi}) \wedge P(D_{fi}, \alpha_{fi}) \tag{6-1}$$

Formula (6-1) applies to both unidirectional and bidirectional dependencies. Additionally, for a bidirectional dependency D, β requires data provided by α through the flow represented by an instance of D. Hence, fulfillment of β may only occur after D has been created which in turn may only be created after α has been created. That is, a Doctor cannot fulfill "Schedule Appointment" until Patient has requested an appointment slot, via an instance of the "Appointment"

dependency, which in turn is only created when a Patient starts executing "Make Appointment". In general:

$$\forall D\ (\ D.depender = \alpha \wedge D.dependee = \beta \wedge BD(D)) \rightarrow (\ Cr(D) \rightarrow O\ Cr\ (\alpha)\ ) \wedge (\ Fi(\beta) \rightarrow O\ Cr\ (D)\ )$$

i.e.

$$P(\alpha_{cr}, D_{cr}) \wedge P(D_{cr}, \beta_{fi}) \tag{6-2}$$

Per section 5.4.3 and Figure 5.6, fulfillment of an elementary electronic dependency D is realized by receiving a designated fulfillment message. An "Appointment" dependency is fulfilled when a Patient receives "Appointment Response" message. In general, the "response received" event D_{rr} of the fulfillment message precedes the fulfillment of the dependency:

$$P(D_{rr}, D_{fi}) \tag{6-3}$$

Additionally, creation of a bidirectional elementary electronic dependency D is realized via a designated request message. Creation of an instance of the "Appointment" dependency triggers sending an "Appointment Request" message. In general, the creation event of D precedes the "request sent" event D_{rs} of the creation message. That is, for a BD(D):

$$P(D_{cr}, D_{rs}) \tag{6-4}$$

The event graph in Figure 5.6 depicts the two pairs in (6-3) and (6-4).

Considering dependency dismissibility, as per section 5.2.2.4, any dependency D is either fulfilled or dismissed. Also its creation condition should imply that the dismissibility condition is not satisfied:

$$Di(D) \rightarrow G \neg Fi(D) \quad \wedge \quad Cr(D) \rightarrow \neg\, Di(D) \tag{6-5}$$

6.2.2 Refinement

AND Refinement: An activity α is AND-refined into sub-activities β_1 through β_n if and only if fulfillment of all sub-activities is required for fulfillment of α. That is:

$$Fi(\alpha) \leftrightarrow Fi(\beta_1) \wedge \ldots \wedge Fi(\beta_i) \wedge \ldots \wedge Fi(\beta_n) \tag{6-6}$$

Since α is not fulfilled until all sub-activities have been fulfilled, the fulfillment event of α may not occur before that of the sub-activities. Hence, $\forall\, i : 1 \leq i \leq n$:

$$\mathbf{Fi}(\alpha) \rightarrow \mathbf{O\ Fi}\ (\beta_i)$$

i.e.

$$\mathbf{P}(\beta_{i\,\mathrm{fi}}, \alpha_{\mathrm{fi}}) \tag{6-7}$$

Also, by definition (Fuxman et al., 2004), α gets instantiated before any of β_i. That is, the creation event of each β_i may only occur after that of α:

$$\mathbf{Cr}(\beta_i) \rightarrow \mathbf{O\ Cr}(\alpha)$$

i.e.

$$\mathbf{P}(\alpha_{\mathrm{cr}}, \beta_{i\,\mathrm{cr}}) \tag{6-8}$$

OR Refinement: An activity α is OR-refined into sub-activities β_1 through β_n if and only if the fulfillment of any of the sub-activities leads to the fulfillment of α. That is:

$$\mathbf{Fi}(\alpha) \leftrightarrow \mathbf{Fi}(\beta_1) \vee \ldots \vee \mathbf{Fi}(\beta_i) \vee \ldots \vee \mathbf{Fi}(\beta_n) \tag{6-9}$$

Fulfillment event of α may not occur before at least one of the sub-activities has been fulfilled. That is, for some *i* where $1 \leq i \leq n$:

$$\mathbf{Fi}(\alpha) \rightarrow \mathbf{O\ Fi}\ (\beta_i) \tag{6-10}$$

Similar to AND refinement, children of an OR-refined activity are only instantiated after their parent has been instantiated, that is formula (6-8) also applies to OR-refinement.

6.2.3 Precedence

A Patient may not start to perform "Visit Doctor" unless they have obtained an appointment by fulfilling "Make Appointment", which is denoted by a precedence link from the latter activity to the former. In general, an activity α must precede β if the fulfillment of α yields information necessary for the creation of β, i.e. for β to be created α must have been fulfilled:

$$\mathbf{Cr(\beta) \rightarrow O\ Fi(\alpha)}$$

i.e.

$$\mathbf{P(\alpha_{fi}, \beta_{cr})} \tag{6-11}$$

As a corollary, since $P(\beta_{cr},\beta_{fi})$:

$$\mathbf{P(\alpha_{fi}, \beta_{cr}) \rightarrow P(\alpha_{fi}, \beta_{fi})} \tag{6-11.a}$$

From 6.11.a, and given transitivity of activity precedence, for any three activities α, β, and χ:

$$\mathbf{P(\alpha_{fi}, \beta_{cr}) \wedge P(\beta_{fi}, \chi_{cr}) \rightarrow P(\alpha_{fi}, \chi_{cr})} \tag{6-11.b}$$

The same semantics apply for dependency precedence. Precedence between dependencies D and E means that E may not be created prior to the fulfillment of D:

$$\mathbf{Cr(E) \rightarrow O\ Fi(D)}$$

i.e.

$$\mathbf{P(D_{fi}, E_{cr})} \tag{6-12}$$

Similar to formulas 6-11.a and 6-11.b, the two following corollaries apply for any three dependencies E, D, and F:

$$\mathbf{P(D_{fi}, E_{cr}) \rightarrow P(D_{fi}, E_{fi})} \tag{6-12.a}$$

$$\mathbf{P(D_{fi}, E_{cr}) \wedge P(E_{fi}, F_{cr}) \rightarrow P(D_{fi}, F_{cr})} \tag{6-12.b}$$

6.2.4 Repetition

We propose a formalization of repetition whose semantics are such that: multiple instances of a repeating activity α may be instantiated in the course of an instance of an interaction, but at most one instance of α exists in any state of the interaction instance. Assume that "Make Appointment" is repeating, i.e. a Patient may need to attempt to execute the activity multiple times until they obtain an appointment. A Patient will not start executing an instance of the activity until they determine that a currently executing instance has failed to obtain an

appointment. In general, in any interaction instance, for a repeating activity α, a yet-to-be-created instance α_{i+1} is never created until it is determined that the currently existing instance α_i will never be fulfilled. Recalling that the temporal logic operator 'G' means "never" and the operator 'F' means "eventually", the relation between instances of α is expressed as:

$$\mathbf{G\ Fi}(\alpha_i) \quad \leftrightarrow \quad \mathbf{F\ Cr}(\alpha_{i+1}) \tag{6-13}$$

Additionally, once a Patient has succeeded in completing an instance of "Make Appointment", they will not perform another instance of the activity in the same instance of the interaction. In general, the fulfillment of α_i means that no further instances of α will be created in the same interaction instance:

$$\mathbf{Fi}(\alpha_i) \quad \rightarrow \quad \mathbf{G\ Cr}(\alpha_{i+1}) \tag{6-14}$$

Specification of repetition propagates through links in a Tropos model. Each instantiation of "Make Appointment" requires a corresponding instantiation of an instance of child activities "Send Request" and "Receive Response" (Figure 5.5) to send a request message and receive a response, respectively. In general, for two activities α and β, where α is parent of β, and the predicate Many() designating repetition as specified in section 5.2.3:

$$\mathbf{Many}(\alpha) \rightarrow \mathbf{Many}(\beta) \tag{6-15}$$

To fulfill an instance of "Make Appointment" an instance of the "Appointment" dependency must be created, so for every instantiation of the former an instance of the latter is also created, from formula (6-2). Thus, the specification of repetition for an activity carries over to any bidirectional dependencies in which it is the depender. That is

$$\forall \mathbf{D}\ (\mathbf{(D.depender} = \alpha) \wedge \mathbf{BD(D)}\ ) \quad \rightarrow \quad \mathbf{Many}(\alpha) \leftrightarrow \mathbf{Many(D)} \tag{6-16}$$

Similarly, to fulfill an instance of "Appointment" a Doctor must instantiate an instance of "Schedule Appointment", so for every instantiation of the former an instance of the latter is also created. Thus the specification of repetition for any dependency carries over to its dependee activity. That is, for any dependency D:

$$\forall \mathbf{D\ (D.dependee} = \beta) \quad \rightarrow \quad \mathbf{Many(D)} \rightarrow \mathbf{Many}(\beta) \tag{6-17}$$

6.3 Generating Messaging Protocol from Tropos Models

By definition, an activity abstracts work performed, by a participant, whose completion leads to fulfillment of the activity. In ACDL this is realized by enclosing work done by the activity in a Sequence whose completion corresponds to activity fulfillment. We have also shown in chapter 5 how dependencies are realized using messaging. Combining these two results with the ordering semantics detailed in the previous section enables us to develop a technique for automatically generating ACDL that realizes a Tropos model.

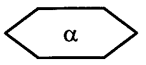
6.3.1 Constraint-Preserving Transformation of Tropos Models

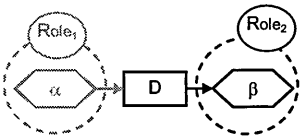
We develop a transform TroposToACDL() that operates on a Tropos model and generates an ACDL messaging protocol compliant with constraints embodied in the model. When applied to a model fragment, TroposToACDL() produces an ACDL block whose execution coincides with the lifecycle of the fragment. To prove the correctness of the transform, we establish a mapping from interaction events, i.e. creation, fulfillment, and message sending events, implied by a Tropos model fragment to execution points in an ACDL block (i.e. beginning and end of ACDL activities). We demonstrate that any execution of the ACDL fragment respects ordering constraints embedded in the corresponding Tropos model fragment as well as the semantics of Tropos constructs, thereby proving that the ACDL fragment is a valid realization of the Tropos model fragment. Figure 6.2 depicts the application of the transform to Tropos diagram fragments. For each Tropos model fragment, we construct a corresponding event graph that captures all interaction events implied by the fragment as well as all ordering constraints between them, as formalized in section 6.2., dashed lines establish a mapping between interaction events and execution points in the corresponding ACDL fragment. In reference to the figure we demonstrate the correctness of the transform:

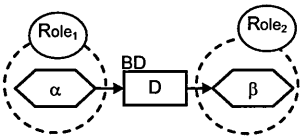
Activity: A Tropos activity α is realized by a Sequence block that matches its lifecycle. Creation of an instance of α corresponds to entering the Sequence block and its fulfillment corresponds to setting a variable denoting its fulfillment. Strict sequential execution of activities within a Sequence guarantee that all work done by α is done only after its instantiation (i.e. after entering

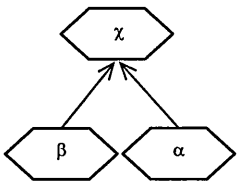
the Sequence) and completed prior to its fulfillment, i.e. prior to executing last activity in the sequence which sets the fulfillment variable. Work done by α includes sending and receiving messages as well as other non-messaging work.

Tropos Model Fragment	TroposToACDL(Model Fragment)

Activity
 
	α_{cr} **Sequence** {
	Perform work embodied by α
	α_{fi} α.fulfilled = true
	}

Unidirectional Dependency
 
	Sequence {
	β_{cr} TroposToACDL(β)
	β_{fi} **If** (β.fulfilled)
	Sequence {
	D_{rr} $Role_2$ **Send** D-Response **To** $Role_1$
	D_{fi} D.fulfilled = true
	}
	}

Bidirectional Dependency
 
	α_{cr} **Sequence** {
	D_{cr} Noop
	D_{rs} $Role_1$ **Send** D-Request **To** $Role_2$
	$\beta_{cr} \rightarrow \beta_{fi} \rightarrow D_{rr} \rightarrow D_{fi}$ TroposToACDL(D as Unidirectional)
	If (D.fulfilled)
	α_{fi} α.fulfilled = true
	}

AND Refinement
 
	χ_{cr} **Sequence** {
	Parallel {
	α_{cr} α_{fi} TroposToACDL(α)
	β_{cr} β_{fi} TroposToACDL(β)
	}
	χ_{fi} χ.fulfilled = true
	}

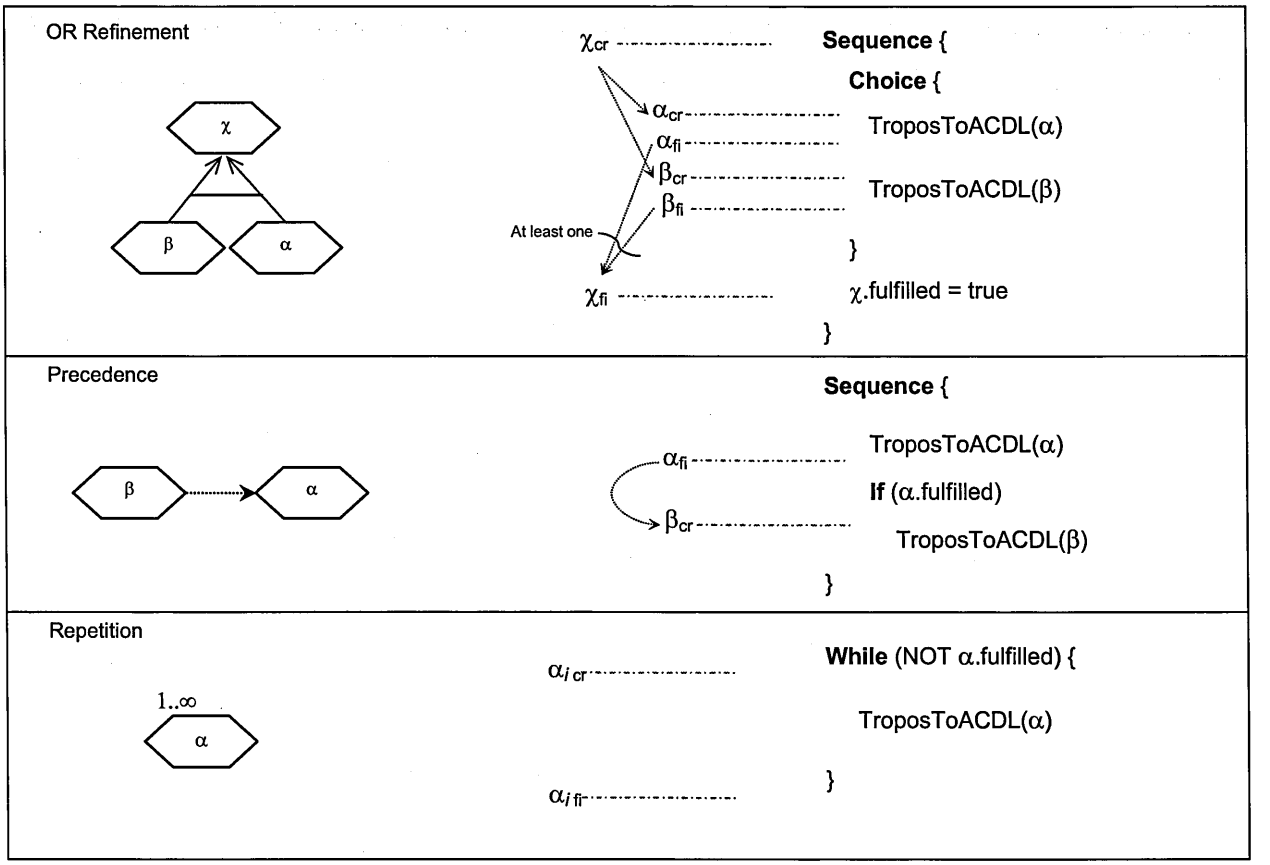


Figure 6.2 Transforming Tropos Diagrams into ACDL Messaging Protocol

Unidirectional Dependency: Fulfillment of the dependency D, from the point of view of the depender, is recorded by setting a corresponding fulfillment variable. Nesting the activity that sets the variable within the inner Sequence guarantees that D is only fulfilled after the fulfillment response message has been sent by dependee and received by the depender (recall that messaging is synchronous), which is consistent with formula (6-3). The outer Sequence along with the If conditional guarantee that message sending and dependency fulfillment may only occur after β has been fulfilled, which is consistent with the constraint $P(\beta_{fi}, D_{fi})$ as specified by formula (6-1). Note that no messaging events correspond to the instantiation of a unidirectional dependency, consistent with the discussion in section 5.2.1.

Bidirectional Dependency: Instantiation of activity α corresponds to the start of the Sequence block and its fulfillment corresponds to the assignment of a fulfillment variable. Having creation

of D correspond to a Noop after the beginning of the Sequence guarantees that the creation of α precedes that of D, consistent with formula (6-2). Also, having the Noop precede sending of the request message is consistent with formula (6-4). Recall that a bidirectional dependency comprises a response part (like a unidirectional dependency) as well as a request part. An ACDL block corresponding to the unidirectional portion of D (i.e. the response part) is added after the request message, which guarantees that the request message precedes the response message, as well as other ordering constraints already guaranteed by the nested block. Finally, following that by the assignment of the fulfillment variable of α guarantees that the fulfillment of D precedes that of α, consistent with (6-1).

AND Refinement: Instantiation of χ corresponds to the start of the Sequence block. Since the beginning of a "Sequence" must be executed before any nested activity, nesting the TroposToACDL() fragment for activities α and β inside an ACDL activity within the "Sequence" guarantees that the creation of χ precedes creation of both α and β, consistent with formula (6-8). The fulfillment event of χ corresponds to setting a corresponding fulfillment variable. The nested Parallel must have completed before the variable is set, which guarantees that the fulfillment of both α and β must occur before that of χ, consistent with formula (6-7). By nesting the blocks of α and β in a "Parallel", they may execute concurrently, consistent with the semantics of "AND" refinement. Since, the "Parallel" block does not complete execution until both activities have completed χ is not fulfilled until both α and β have been fulfilled, consistent with formula (6-6).

OR Refinement: Similar to AND-refinement, instantiation of χ corresponds to the start of the Sequence block which guarantees that the creation of χ precedes creation of both α and β nested within, consistent with formula (6-8). The fulfillment event of χ maps to setting a corresponding fulfillment variable. The nested Choice must have completed before the variable is set, which guarantees that the fulfillment of at least one of α and β must occur prior to that of χ, consistent with formula (6-9). Since, the "Choice" block does not complete execution until at least one of

the enclosed activities have completed, χ is not fulfilled until either α or β has been fulfilled, consistent with formula (6-10).

Precedence: An ACDL Sequence guarantees that an enclosed activity is not executed until the preceding activity has completed. Preceding the block for β with the block for α and guarding with the If guarantees that creation of β occurs only after α has been fulfilled, consistent with formula (6-11).

Repetition: Recall that an activity nested in a While construct may execute multiple times. A repeating activity is realized by a While block, where each execution of the ACDL activity within the While construct corresponds to the lifecycle of an instance of α. Since only one execution of the activity nested within the While is in progress at a time, an execution of the nested ACDL instance corresponding to an instance of α does not start except if the previous one, if any, has completed and failed to fulfill α, consistent with formula (6-13). Once α has been fulfilled, the While block exits and no more execution of the nested activity will occur, thus no more instances of α will be created, consistent with formula (6-14). Since prior to the While block α has not been executed, α is initially not fulfilled, so the While block is guaranteed to execute at least once.

6.3.2 Traversing a Tropos Model Graph

A CLG can be expressed as a labeled graph where the nodes are activities and dependencies while the edges are links between them. Recursive application of the transform to diagram fragments as in Figure 6.2 implicitly defines a traversal of this graph. The traversal serves to compose generated protocol fragments into a messaging protocol that realizes a CLG. Consider an example application of the transform to an activity "A" involving outgoing dependency (i.e. a dependency in which "A" is the depender), AND-refinement, incoming dependency (i.e. a dependency in which "A" is the dependee), and precedence in Figure 6.3. As in Figure 6.2, we create an event graph for the depicted CLG, associate interaction events with execution points in the corresponding ACDL, and annotate the ACDL with the interaction events.

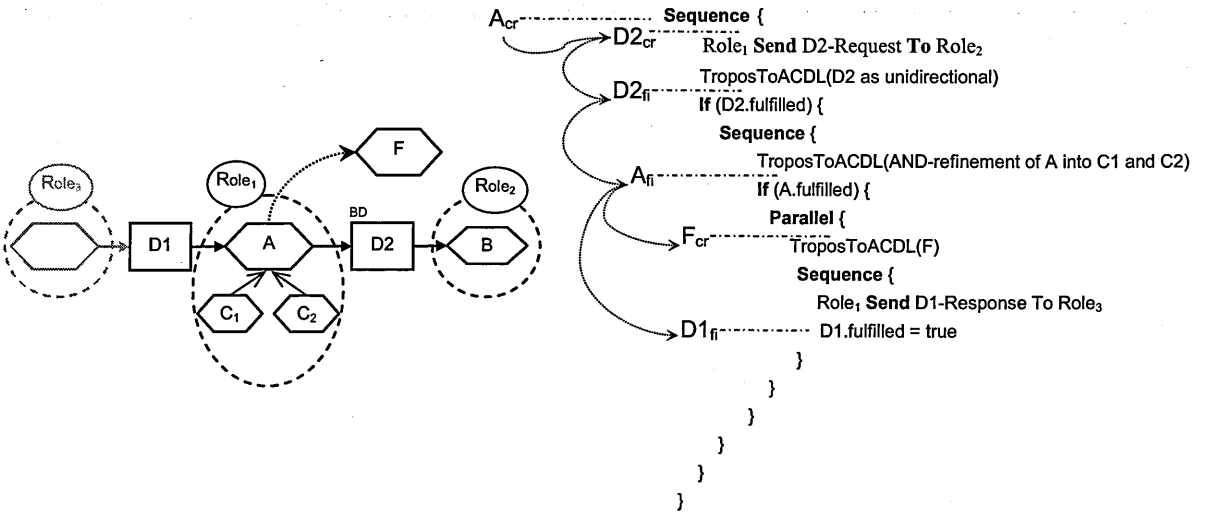


Figure 6.3 Traversing a CLG to generate ACDL

This example demonstrates that the composition of ACDL fragments can be automated. Consider events required for fulfillment of activity A and events that depend on its fulfillment:

1. Events required for A_{fi}: For activity A to be fulfilled, all its child activities as well as the dependency D2 must have been fulfilled. This requires composition of the "Bidirectional Dependency" and "AND Refinement" fragments from Figure 6.2. This is accomplished by nesting the "AND Refinement" transform of A into C1 and C2 inside the Sequence that follows the fulfillment of D2, as per the "Bidirectional Dependency" transform.

2. Events that require A_{fi}: Fulfillment of A is required for the fulfillment of the dependency D1 as well as for the activity F. According to the "Unidirectional Dependency" fragment and the "Precedence" fragment this is accomplished by conditioning the D1-Response Sequence and the execution of F on the fulfillment of A, respectively. This is manifested by nesting both the D1-Response Sequence and the execution of F in a Parallel that is conditioned on the fulfillment of A.

The correctness of the generated ACDL can be argued in the same manner as was done for the constituent fragments, i.e. by demonstrating that the execution of the generated ACDL enforces the ordering constraints between the interaction events. For example:

- Creation of activity A precedes creation of bidirectional dependency D2 as well as sending of the D2-request message by virtue of enclosing the latter within the Sequence corresponding to A.

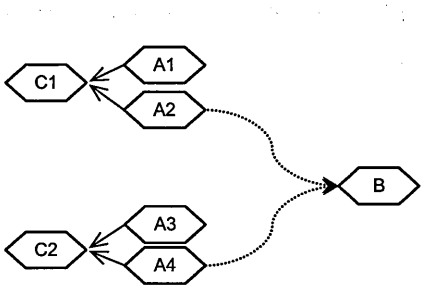
- Activity A is only fulfilled when both D2 has been fulfilled and all its sub-activities have been fulfilled, by virtue of enclosing the response part of D2 within the outer Sequence as well as enclosing the refinement into C1 and C2 in the middle Sequence.

- Precedence between A and F is realized by enclosing the block for F inside the middle Sequence after A has been fulfilled.

- Similarly, D1 is not fulfilled except after A has been fulfilled by virtue of enclosing in a nested Sequence within the middle Sequence.

We implemented an automated tool that traverses a Tropos model graph and generates an ACDL protocol that realizes the model. Our tool composes ACDL fragments that realize Tropos model fragments while attempting to exploit parallelism in a Tropos model. For example, execution of activity F and sending of fulfillment response of dependency D1 are allowed to execute concurrently by enclosing them in a "Parallel". The tool is publicly available for download at http://www.sourceforge.net/projects/chreq.

A choreographed interaction starts when an "initiator" activity starts executing (Ross-Talbot & Fletcher, 2006). For example, in the medical interaction, "Get Authorized" is the initiator activity. To generate an ACDL protocol from a CLG diagram, we apply our transform to the initiator activity. Two issues arise with traversal of the CLG graph:




```
Parallel {
  Sequence {
    Parallel {
      TroposToACDL(A1)
      TroposToACDL(A2)
    }
    C1.fulfilled = true
  }
  Sequence {
    Parallel {
      TroposToACDL(A3)
      TroposToACDL(A4)
    }
    C2.fulfilled = true
  }
  While ¬(A2.fulfilled ∧ A4.fulfilled)
    NOOP
  TroposToACDL(B)
}
```

Figure 6.4 Example of converting Tropos diagram graph into ACDL tree

1. An ACDL specification is strictly a tree, whereas a Tropos diagram is generally a graph. Thus, the traversal algorithm is tasked with converting a graph into a tree. Where it is not possible to represent a graph edge as a tree edge, alternate forms of the transform are employed. Problematic edges of the graph are represented instead as ACDL conditions. A notable example is an activity with two incoming precedence links as in Figure 6.4.

2. Combining the local models of participants may yield an interaction that cannot be realized as a messaging protocol (Kazhamiakin & Pistore, 2006). The resulting combined model may exhibit deadlocks, e.g. in the form of cyclical dependencies. A form of topological sort (Cormen et al., 2002) is utilized to ensure that any node is only processed when all its prerequisites have been processed. By using topological sort such anomalies are detected and reported by our tool.

6.4 From ACDL to WS-CDL

To ground our approach in service-oriented standards, we provide a path for generating a skeletal WS-CDL description from an ACDL protocol. Following the XML syntax of WS-CDL (Kavantzas et al., 2005), a WS-CDL description is composed of three main parts:

- Package-level definitions: which are meta-level definitions for interacting role types, relationship types between them, and structure of messages they exchange.

- Choreography-level definitions: which are instance-level definitions of variable and relationships between instances of role types.
- Messaging Specification: specifies valid messaging sequences for the choreographed interaction between role instances.

6.4.1 Generating WS-CDL Package-Level Definitions

Package-level definitions in WS-CDL are all optional elements. However, to generate a meaningful and useful choreography, we need to generate at least three elements:

1. Role Types: Choreographed role types correspond directly to roles on either end of ACDL "Send" activities. Hence, we generate a "roleType" element for every role in an ACDL "Send" activity, eliminating duplicates. Each role type must have at least one "behavior" element, one such element is generated for each role to aggregate its observable behavior.

2. Relationship Types: A relationship type specifies that a pair of roles exchange messages. By examining all "Send" activities, we enumerate pairs of roles that exchange messages. For every unordered pair of roles, a "relationshipType" element is generated.

3. Information Types: Data types define the structure of message contents. An "informationType" element is generated for every type of message in an ACDL protocol, and is named after the message type.

6.4.2 Generating WS-CDL Choreography-Level Definitions

To obtain a WS-CDL "choreography" element, we generate the required elements:

1. Relationships: For every relationship type defined at the package level, a "relationship" element that instantiates the corresponding type.
2. Variables: a "variableDefinitions" element that defines two types of variables:
    a. Variables that instantiate each message in the "informationType" element.
    b. For every CLG activity, a Boolean variable that denotes its fulfillment.

6.4.3 Generating WS-CDL Messaging Specification

To generate a WS-CDL messaging specification from ACDL we define a transform AcdlToWSCDL() that maps ACDL control flow and messaging constructs into equivalent WS-CDL constructs (Table 6.1). "Parallel", "Sequence", and "Choice" constructs are trivially translated into corresponding WS-CDL constructs. For conditional activities and iteration, a WS-CDL "workunit" is constructed using the condition of the ACDL construct. Finally, an ACDL "Send" is translated into WS-CDL primitive activity, an "interaction".

Table 6.1 Transforming ACDL constructs to WS-CDL constructs

ACDL Construct	AcdlToWSCDL Applied to ACDL Construct
Sequence Activity*	**<sequence>** AcdlToWSCDL(Activity*) **</sequence>**
Parallel Activity*	**<parallel>** AcdlToWSCDL(Activity*) **</parallel>**
While Condition Activity	**<workunit guard**="XPath-equivalent of Condition" **repeat**="XPath-equivalent of 'Condition'" **block**="true"> AcdlToWSCDL(Activity) **</workunit>**
If Condition Activity$_1$ **Else** Activity$_2$	**<workunit guard**="XPath-equivalent of Condition" > AcdlToWSCDL(Activity$_1$) **</workunit>** **<workunit guard**="XPath-equivalent of negated Condition" > AcdlToWSCDL(Activity$_2$) **</workunit>**
R_1 **Send** Message **To** R_2	**<interaction name**="Descriptive name for interaction"> **<participate** **relationshipType**="WS-CDL relation between R_1 and R_2" **fromRoleTypeRef**="WS-CDL role for R_1" **toRoleTypeRef**="WS-CDL role for R_2"> **<exchange** **name**="Descriptive name for exchange" **informationType**="type of 'Message'" **action**="Direction of 'Send'"> **<send variable**="variable for 'Message'"/> **<receive variable**="variable for 'Message'"/> **</exchange>** **<interaction>**

6.4.4 Example Generation of WS-CDL from ACDL

To demonstrate the generation of WS-CDL from ACDL, we apply AcdlToWSCDL() transform to the following ACDL fragment to WS-CDL:

Patient **Send** Appointment Request **To** Doctor

Doctor **Send** Appointment Response **To** Patient

The resulting WS-CDL is shown in Figure 6.5.

```
<package name="PatientDoctorChoreography">

  <informationType name="AppointmentRequestType"/>
  <informationType name="AppointmentResponseType"/>

  <roleType name="Doctor">
    <behavior name="DoctorForPatient"/>
  </roleType>
  <roleType name="Patient">
    <behavior name="PatientOfDoctor"/>
  </roleType>

  <relationshipType name="PatientDoctorRelationship">
    <roleType typeRef="tns:Doctor" behavior="tns:DoctorForPatient"/>
    <roleType typeRef="tns:Patient" behavior="tns:"PatientOfDoctor"/>
  </relationshipType>

  <choreography name="PatientDoctorChoreography">
    <relationship type="tns:PatientDoctorRelationship"/>

    <variableDefinitions>
      <variable name="AppointmentRequest" informationType="tns:AppointmentRequestType"/>
      <variable name="AppointmentResponse" informationType="tns:AppointmentResponseType"/>
    </variableDefinitions>

    <sequence>
      <interaction name"GetAppointment">
        <participate relationshipType="tns:PatientDoctorRelationship"
                     fromRoleTypeRef="tns:Patient" toRoleTypeRef="tns:Doctor"/>
        <exchange name="request" informationType="tns:AppointmentRequestType" action="request">
          <send variable=cdl:getVariable('tns:appointmentRequest','','')"/>
          <receive variable=cdl:getVariable('tns:appointmentRequest','','')"/>
        </exchange>
      </interaction>
      <interaction name"GetAppointment">
        <participate relationshipType="tns:PatientDoctorRelationship"
                     fromRoleTypeRef="tns:Patient" toRoleTypeRef="tns:Doctor"/>
        <exchange name="response" informationType="tns:AppointmentResponseType" action="respond">
          <send variable=cdl:getVariable('tns:appointmentResponse','','')"/>
          <receive variable=cdl:getVariable('tns:appointmentResponse','','')"/>
        </exchange>
      </interaction>
    </sequence>

  </choreography>
</package>
```

Package-level

Choreography level

Messaging Specification

Figure 6.5 WS-CDL Generated from example ACDL snippet

The verbosity of the resulting WS-CDL justifies using ACDL to present our work. Moreover, the WS-CDL document is just a skeleton which needs to be augmented by adding further design details. In particular, the following information is omitted from the document:

- Data type structure describing fields of each message.
- Message correlation tokens.
- References to service interfaces, e.g. operations declared in WSDL documents.
- XML Namespace declarations (Bray et al., 2009).

6.5 Chapter Summary

This chapter elaborated on our technique for systematic derivation of choreographed messaging specification from Tropos models. To facilitate the derivation, we explicated ordering semantics embedded in Tropos models. We devised a transform that operates on a Tropos model to generate a messaging specification consistent with constraints captured in the model. Our implementation of the transform traverses a Tropos model and generates a messaging protocol in an Abstract CDL (ACDL). We demonstrated how skeletal choreography description in standard CDLs, such as WS-CDL, can be generated from ACDL. Deriving choreographed messaging from requirements models is central to our approach to adaptation. It enables us to perform adaptations at the level of requirements models, which we detail in the next chapter.

Chapter 7. Guidance for Disciplined Choreography Adaptation

This chapter reports on the third contribution of this thesis: providing guidance for disciplined adaptation of a choreographed interaction specification. Prior to presenting our adaptation framework, we detail the needs that any such framework should address. Next, we outline a metamodel which we employ to guarantee that adaptation of a CLG produces a structurally valid specification of an interaction. We build a catalogue of operations for adapting the global and local views of a CLG and propagating changes systematically between them. The catalogue provides fine-grained guidance on adapting elements of a CLG, navigating the space of adaptation alternatives, and producing a set of consistent views. Finally, we weave these techniques into a process flow that guides forging an agreement between stakeholders on an adapted interaction specification.

7.1 Characteristics of an Interaction Adaptation Framework

From the discussion in sections 2.4 and 3.4, a framework for adapting a choreographed interaction specification has to address the needs of several stakeholders each with a different view on the interaction. First and foremost, an adaptation framework must maintain the structural and semantic validity of the global and local views. Additionally, the framework must keep views consistent, by propagating changes between them. By maintaining consistency between views the framework enables stakeholders to reach an agreement on an adapted specification. We elaborate these needs and what is entailed in satisfying each of them.

7.1.1 Adaptation of a Local View

Adaptation of a local view must yield a valid model that complies with structural and semantic constraints on activities, their refinement, precedence, and lifecycle conditions. For each type of change to a local view, guidance is required to ensure that adaptation produces a valid view. The space of alternatives for performing a certain adaptation may be large. Searching

this space for an adaptation that achieves participant's business goals would benefit from guidance on enumerating alternatives.

7.1.2 Adaptation of the Global View

Adaptation of the global view must yield a valid model that complies with structural and semantic constraints on dependencies, their precedence, and lifecycle conditions. Similar to local view adaptation, for each type of change to the global view, guidance is required to ensure that adaptation produces a valid view. Also similarly, adaptation of the global view benefits from guidance on searching the space of adaptation alternatives.

7.1.3 Change Propagation from a Local View to the Global View

Activities in a local view may delegate responsibility via dependencies to other participants. This linkage implies that adaptation of a local view, e.g. adding activities, may impact the global view. Additionally, per section 6.2, adapting constraints on activity execution may impact lifecycle conditions of related dependencies. To keep the specification of observable behavior of a participant consistent with that of their internal execution, it is necessary to propagate changes from their local view to the global view. Therefore, for every type of change in a local view, guidance is required on determining impact of the change on the global view.

7.1.4 Change Propagation form the Global View to a Local View

Change propagation between local views and the global view is bidirectional; changes to the global view may impact local views. Changes in dependencies, their precedence, and lifecycle conditions may impact activities in one or more local views. To keep the specification of internal participant behavior consistent with the specification of observable behavior, it is necessary to propagate changes from the global view to local views. Therefore, for every type of change in the global view guidance is required on determining impact of the change on each local view.

7.1.5 Agreement on Interaction Specification

By supporting bidirectional change propagation, from the global view to local views and vice versa, an adaptation framework helps maintain a set of local views consistent with the global

view that ties them together. A global view consistent with a set of local views constitutes a valid interaction specification and designates agreement between stakeholders. An adaptation framework should steer stakeholders towards forging such an agreement. Guidance is required for determining whether such an agreement is possible and also for determining when an agreement is reached.

7.2 Overview of Our Interaction Adaptation Framework

Having outlined the needs that any framework for interaction adaptation has to satisfy, we outline how our proposed framework satisfies these needs. Our adaptation framework guarantees the validity of an adapted interaction specification by employing a metamodel that constrains the structure of valid models. Based on the metamodel, we build a catalogue of operations for adapting the global and local views of an interaction specification. We outline these operations as well as operations for change propagation between views. The metamodel and the operations guide the obtainment of a valid interaction specification composed of a consistent set of views.

7.2.1 Metamodel for Interaction Specification

To ensure structural validity of an interaction specification, we need to detail constraints that govern the structure of a valid CLG model. The Tropos metamodel (Giunchiglia et al., 2002) does not fully support capturing the structure of a CLG diagram, which was introduced later (Fuxman et al., 2004), neither do extensions of the Tropos metamodel (Susi et al., 2005). In particular, although the original metamodel supports the specification of depender and dependee roles in a dependency, it does not allow the specification of depender and dependee activities at each end. We thus put forward a metamodel that captures structural constraints governing CLG models (Figure 7.1). Using the UML metamodel syntax (Booch et al., 1999), each model element is represented as a box with relations between them, as well as cardinality of the relations, specified on connecting lines. A model is said to be structurally valid if it conforms to constraints captured in the metamodel. For instance, the metamodel stipulates that a creation and a fulfillment condition must be specified for any activity and that an activity is assigned to exactly one role. Elements of the metamodel are annotated with semantic constraints, in OCL

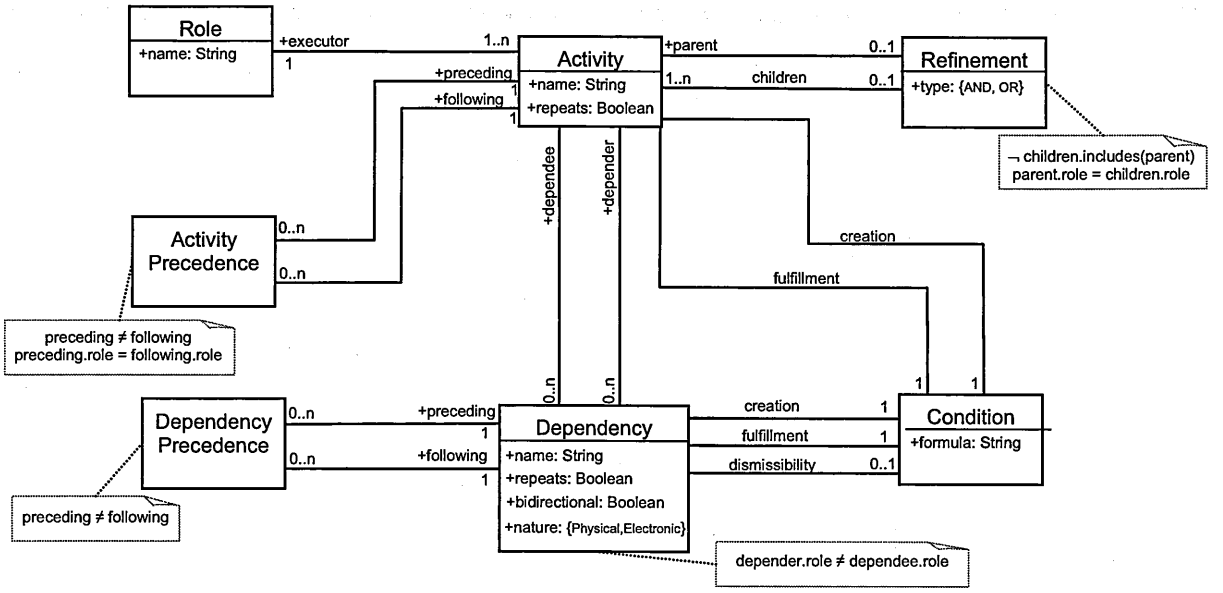


Figure 7.1 Metamodel for interaction specification

(OMG-OCL, 2010), to further constrain the structure of a valid model. For example, a valid refinement of an activity requires that the parent and children activities belong to the local view of the same role and that a parent may not be the child of itself.

Although metamodel conformance is necessary for a structurally valid model, it is not sufficient for a meaningful one. A meaningful specification must be deadlock-free, i.e. it must yield a cycle-free interaction event graph. In a complex model, a pan-view cycle involving a combination of precedence links, dependencies, as well as refinement links may crop up. Our automated tool, introduced in section 6.3.2, supports checking of the validity of a CLG model and detecting cycles, thereby assisting with the adaptation process. More subtly, the constraints formalized in chapter 6 which bestow meaning on model elements must hold in any meaningful specification. For example, fulfillment conditions of child activities must be consistent with fulfillment condition of their parent, as specified in section 6.2.2. Our catalogue of adaptation operations guides the preservation of these semantic invariants throughout the adaptation process.

7.2.2 Catalogue of Operations for Adaptation and Change Propagation

Adaptation of a CLG model is achieved by performing "adaptation operations". An adaptation operation involves adding/removing an element to/from a Tropos model (Krishna et al., 2004). By capturing model elements and their relations, the metamodel allows us to enumerate potential adaptation operations. We construct a catalogue of operations (Figure 7.2) to guide step-by-step adaptation of an interaction specification and guarantee the validity of the outcome.

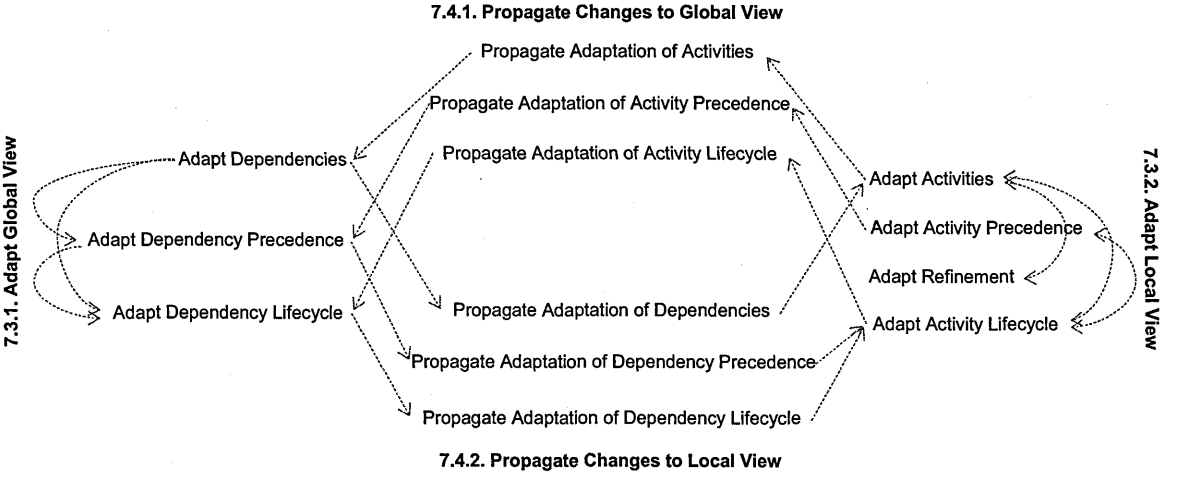


Figure 7.2 Catalogue of Adaptation Operations and Change Propagation Operations

The catalog is not merely a listing of operations as it helps:

- Associate a business meaning with each operation. For example, adapting the creation condition of a dependency constrains the depender's ability to request its fulfillment.
- Enumerate options for achieving what is intended from each operation. For instance, an operation whose intention is to ensure that "the execution of one activity completes before another activity starts" can be performed either by adding a direct precedence link from the first activity to the second or alternatively via indirect means. To ease the navigation of the space of alternative ways for performing an operation, where applicable, the catalogue guides the enumeration of alternatives.

- List structural constraints that govern an operation and means to maintain conformance of a model being adapted to the Tropos metamodel. For instance, to add an activity, a fulfillment condition and creation condition must also be specified.

- Specify which semantic invariants (from section 6.2) are perturbed by each operation and means to maintain satisfaction of these invariants. For instance, consistency of the fulfillment conditions of child activities with that of their parent may be affected by adapting the latter.

- Suggest further operations to propagate changes and restore consistency to the interaction specification. For instance, adding precedence between dependencies (i.e. in the global view) may require adapting the lifecycle conditions of associated activities in a manner described by the catalog.

The last point is central to guiding orderly progression of the adaptation process. We employ links in the metamodel as well as semantic invariants to drive consideration of further changes that follow an adaptation operation, whether in the same view or another view. On the one hand, adding an activity requires specifying its lifecycle conditions, and may also entail refining it into other activities in the same local view. On the other hand, adding an activity to a local view may also involve linking it to new or existing dependencies in the global view. Guidance on the progression of the adaptation process is represented by the arrows in Figure 7.2 and is backed by cross-referencing between operations in the catalog as detailed in sections 7.3 and 7.4.

The next section 7.3 catalogues the adaptation operations for the global and local view, and section 7.4 catalogues operations for guiding change propagation between views. We illustrate the guidance provided by the operations using the medical example introduced in section 2.2.2, as well as suggested adaptation thereof. Recall that an MP needed to ensure that they cover treatment expenses only for eligible patients. To achieve this need an MP requires a Doctor to collect a Patient's Medical Plan ID (MPID) and provide it to the MP to check its validity. Dependencies, activities, and links used to incorporate this need into the CLG of the medical example are depicted in Figure 7.3, as well as their relations to elements in the original CLG.

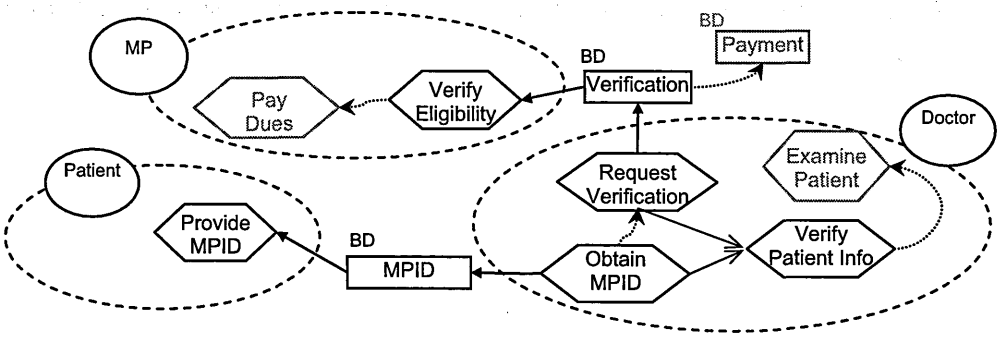


Figure 7.3 Example adaptation of the medical example CLG to verify Patient eligibility

7.3 Guiding Adaptation Operations of Interaction Specification

Using the metamodel we catalogue adaptation operations of each view of an interaction specification. Adaptation operations of local views manipulate activities, their refinement, precedence, and their lifecycle conditions whereas those of the global view manipulate dependencies, their precedence, and conditions. Structural constraints imposed by the metamodel dictate what elements must be manipulated by each operation. Further to the structural guidance inferred from the metamodel, we provide finer-grained guidance for each operation in order to preserve semantic invariants. Where relevant, we also employ those semantics to guide the enumeration of alternative ways of performing each operation.

7.3.1 Guiding Adaptation Operations of the Global View

7.3.1.1 Adapt Dependencies

Adapting dependencies involves adding or removing a dependency. Adding a dependency requires specifying:

- A depender role, which needs the fulfillment of the dependency, and a dependee role, to which the fulfillment responsibility is delegated.
- Identifying a depender activity and a dependee activity (sections 7.4.2.1 and 7.3.2.1).

- Creation conditions, which are satisfied when the depender desires the fulfillment of the dependency, and fulfillment conditions, which are satisfied when the dependee has fulfilled the delegated responsibility (section 7.3.1.3).

- Whether the dependency is physical or electronic.

- Whether the dependency is bidirectional. A dependency is bidirectional if the dependee cannot fulfill their responsibility unless the depender has supplied some information.

- Dismissibility condition and repetition, if any (section 7.3.1.3).

Assume the need to protect an MP from abuse of treatment coverage by a Patient, introduced in section 2.4, is realized by requiring a Doctor to request verification of Patient MPID from an MP prior to billing. This realization involves delegation of "Verification" responsibility from a Doctor to an MP, via a dependency in which Doctor is depender and MP is dependee. Having identified the depender and dependee, the choice of the depender and dependee activities is a concern of the local views and is thus left to the depender and dependee roles to decide, respectively. The "Verification" dependency requires that a Doctor provides an MPID to be verified to an MP, and is thus annotated as bidirectional. Assuming that the verification is to occur via messaging, the dependency is marked as being electronic.

Removing a dependency from a CLG requires removing its links to the depender and dependee activities as well as removing all precedence links, incoming and outgoing. Incoming and outgoing precedence links of a dependency D may contribute to establishing a transitive precedence relation between other dependencies. That is if a dependency E precedes D and also D precedes F, then E transitively precedes F. Removing D and its precedence links requires adding a precedence link between E and F to maintain their precedence relation (Figure 7.4), as well as potentially other adaptation to precedence, as discussed in the next section.

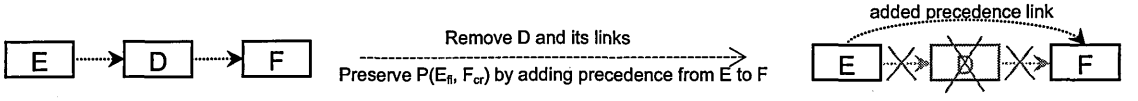


Figure 7.4 Preserving precedence after removing a dependency

7.3.1.2 Adapt Dependency Precedence

Contextual constraints may dictate an order in which roles are required to fulfill their responsibilities towards each other. Dependency precedence helps capture these constraints by specifying a partial order over interaction events, from a global point of view. A precedence relation between two dependencies D and E specifies that creation of E must not occur prior to fulfillment of D. Precedence between two dependencies can be established either directly or transitively. In its direct form, a precedence relation between D and E can be established by adding a precedence link from D to E. Alternatively, per formula (6-12) and its two corollaries, precedence between D and E may be specified by adding a precedence link from D to any dependency F that, either directly or transitively, precedes E (Figure 7.5). The set from which F is chosen is enumerated by identifying all dependencies satisfying $P(F_{fi}, E_{cr})$.

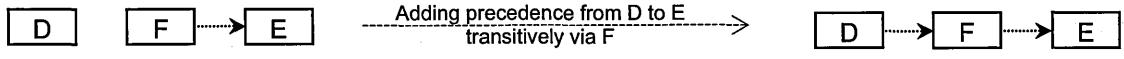


Figure 7.5 Adding precedence between two dependencies transitively via a third dependency

Consider the need to specify that a Patient is required to obtain an "Authorization" prior to attempting to fulfill "Appear for Exam" dependency. Although this constraint is enforced indirectly via precedence links in local models, it can be captured in the global model by adding a direct precedence link from the "Authorization" dependency to the "Appear for Exam" dependency. Alternatively, since "Authorization" already precedes "Appointment", adding a precedence link from "Appointment" to "Appear for Exam" establishes transitive precedence from "Authorization" to "Appear for Exam".

Conversely, eliminating a precedence link from D to E eliminates all direct and transitive precedence implied by this link. Assume that E precedes a third dependency F, thereby establishing transitive precedence from D to F. In case it is desired to maintain precedence from D to F after removal of precedence between D and E, a new precedence relation between D and F has to be established in the manner described above (Figure 7.6).

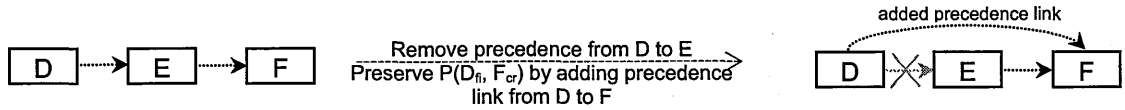


Figure 7.6 Preserving transitive precedence after removing a precedence link

Dependency precedence is a partial ordering relation, i.e. it does not allow precedence cycles. To respect this invariant, any adaptation of dependency precedence must not create cycles, i.e. must not yield a transitive precedence where a dependency D precedes itself, which yields an invalid pair $P(D_{fi}, D_{cr})$.

7.3.1.3 Adapt Dependency Lifecycle

Adapting a dependency lifecycle condition involves adapting its creation, fulfillment, and/or dismissibility conditions as well as its repetition. Guidance on these operations applies whether they are performed separately or as part of adding a new dependency.

Creation: Recall from 4.3.1.1 that a dependency is instantiated at the point where the depender activity starts desiring its fulfillment. Adapting the creation condition can thus be used to restrict the behavior of the depender in a dependency. For example, assume that an MP is allowed to suspend the medical plan for a Patient who violates its terms. Suspending the plan involves temporarily disallowing a Patient to schedule appointments with any Doctor. This constraint can be enforced by adapting the creation condition of the "Appointment" dependency to assert that the Patient's medical plan is active (i.e. not suspended).

Adapting a creation condition is also a way for specifying information that a depender must provide for a bidirectional dependency to be instantiated. The creation condition is constructed in a way that asserts the availability of this information. For the "Verification" dependency (Figure 7.3), a Doctor is required to provide to an MP the MPID to be verified. This constraint can be incorporated directly, by adapting the creation condition of "Verification" to include the "availability of an MPID" as a necessary condition, or indirectly via dependency precedence by adding a precedence link from the "MPID" dependency to the "Verification" dependency, following the guidance provided in section 7.3.1.2.

Fulfillment: Adapting the fulfillment condition of a dependency changes the responsibility that a dependee has to fulfill. Recall from section 5.2.1 that dependency fulfillment is contingent upon availability of required fulfillment information. The fulfillment condition of a dependency is constructed such that it asserts that the fulfillment information has been made available by the

dependee. For example, assume that a Doctor is to be required to provide a Confirmation Number (CN) to the Patient whenever she schedules an appointment. This can be enforced by adapting the fulfillment condition of the "Appointment" dependency to assert that the CN is available. Since a dependency is only fulfilled when the dependee activity has been fulfilled, from (6-1), the constraint can alternatively be enforced by adapting the fulfillment condition of the dependee activity to assert the availability of the CN (sections 7.3.2.4 and 7.4.1.2).

Since a precedence relation relates the creation condition of any dependency to its fulfillment, adaptation of the creation condition may affect the fulfillment condition. For instance, stipulating that a Patient may not request an appointment unless they have an active medical plan (at the time of the request) also implies that, to obtain an appointment, a Patient must have had an active plan (i.e. when they requested an appointment). In general, if the creation condition of a dependency implies that a condition Φ held at some point in the past, the fulfillment condition must also imply that Φ held at some point in the past. That is: $(Cr(D) \rightarrow O\ \Phi) \rightarrow (Fi(D) \rightarrow O\ \Phi)$, and any adaptation of the creation or fulfillment conditions must satisfy this invariant.

Dismissibility: Recall, from section 5.2.2.4 that the dismissibility condition of a dependency specifies circumstances under which a dependee is allowed to dismiss the dependency, i.e. choose to not fulfill it. Adapting a dismissibility condition guards interests of a dependee against a depender who instantiates a dependency in a situation where the creation condition is not satisfied. For instance, to enforce that no payment is made prior to the verification of a Patient MPID, an MP may stipulate that the "Payment" dependency will be dismissed if the Patient MPID has not been verified. This privilege given to the dependee must not conflict with their obligation to fulfill dependencies created when the creation condition indeed held. That is, any adaptation of the dismissibility condition must satisfy the invariant specified by (6-5). Conversely, any adaptation of creation or fulfillment conditions must satisfy the same invariant. Making a dependency dismissible is typically coupled with annotating it as being repeating, to allow the depender to re-attempt fulfilling the dependency.

Repetition: Recall from section 5.2.2.2 that for a repeating dependency to be fulfilled, multiple instantiations of the dependency may be required in one instance of an interaction. Adapting a

dependency to be repeating thus allows a depender to re-attempt fulfillment of a dependency if one or more instances fail, typically due to dismissal by the dependee. For example, annotating the "Appointment" dependency as being repeating allows a Patient to re-request an appointment if one or more earlier requests were dismissed by a Doctor, e.g. due to unavailability of appointment slots.

7.3.2 Guiding Adaptation Operations of Local View

7.3.2.1 Adapt Activities

Adapting activities involves either adding or removing an activity. An activity represents work done by a participant as part of an interaction. To represent an additional task that a participant is required to perform, an activity is added to their local view. For instance, to ask the MP to verify an MPID, a Doctor adapts their local view by adding a "Request Verification" activity. Adding an activity requires at least specifying conditions under which it may start performing its work, i.e. its creation conditions, and criteria for completing the work, i.e. its fulfillment conditions (section 7.3.2.4).

Removing an activity α involves removing all its precedence links. To preserve transitive dependencies that α was part of, it may be necessary to add new precedence links (in the same manner as described in section 7.3.1.1). If the intention of removal is to omit all work done by α, then all its child activities are also removed. Otherwise, activity refinement in the local view of α needs rework, which may involve reattaching orphaned children of α to another parent (section 7.3.2.2).

7.3.2.2 Adapt Refinement

To improve modularity of a local view, newly added activities or newly orphaned activities, i.e. those whose parent was removed, should be logically grouped under a parent, new or existing. Refinement serves to group together activities that share the same creation condition and collective fulfillment condition. For any two activities α and β where β is a sub-activity of α, adaptation of activity AND/OR refinement must respect these invariants:

- Both activities belong to the same local view.
- α must not be a sub-activity of β, directly or transitively.
- β must not be a sub-activity of any other activity.
- A necessary creation condition of α is also a necessary for creation of β (6-8).

AND-refinement: AND-refinement of an activity α breaks down work done by α into finer-grained parts whose completion completes the work of α, i.e. completion of all the parts is equivalent to completion of the whole. Any adaptation of AND refinement must preserve the invariant specified by (6-6) where α is completed if and only if all its children have completed. Consider the work that a Doctor has to perform to verify a Patient's information. A Doctor has to collect information from a Patient AND verify it with the MP. This work is represented by two activities in the Doctor's local view: "Obtain MPID", whose fulfillment stipulates that Patient MPID has been obtained, i.e. Available(Patient.MPID), and "Request Verification", whose fulfillment stipulates that Patient information has been verified, i.e. Verified(Patient.MPID). For modularity, both activities are made sub-activities of a new activity "Verify Patient Info" (Figure 7.3), whose fulfillment is the conjunction of the fulfillment conditions of the two sub-activities i.e., Available(Patient.MPID) $\wedge$ Verified(Patient.MPID). This modularity allows the stipulation of a new creation condition on both activities by adapting the creation condition of their parent, from formula (6-8).

OR-refinement: In addition to the four invariants that apply to both types of refinement, any adaptation to an OR-refinement must preserve an additional invariant. As OR-refinement represents alternatives for performing the work done by an activity α, completion of any of the sub-activities must lead to fulfillment of the parent, per (6-9). Thus, fulfillment condition of each sub-activity of an OR-refined activity must imply fulfillment of its parent, as per (6-10).

7.3.2.3 Adapt Activity Precedence

Local business policies may dictate an order in which a participant performs their activities. Activity precedence helps capture these constraints by specifying a partial order over interaction events from a participant's local point of view. A precedence relation between two activities α and β specifies that β may not start executing unless α has completed. Precedence between two

activities can be established either directly or transitively. Adding a precedence link from α to β establishes a direct precedence between them. Alternatively, precedence between α and β can be specified by adding a precedence link from α to any activity χ that, either directly or transitively, precedes β (Figure 7.7). The set of activities to which χ belongs is enumerated by identifying all activities satisfying $P(\chi_{fi}, \beta_{cr})$.

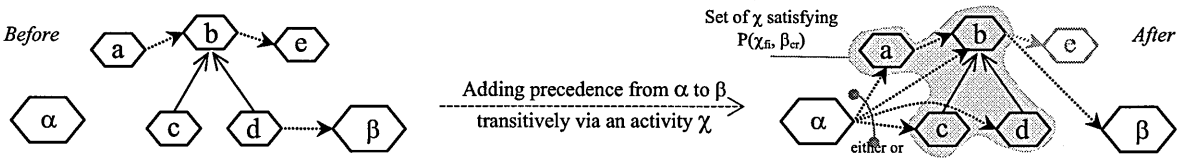


Figure 7.7 Adding precedence between two activities transitively via a third activity

Consider the Doctor's need to specify that completion of "Verify Patient Info" must precede "Collect Payment". This need can be specified directly by adding a precedence between the two activities, or transitively, by adding a precedence from "Verify Patient Info" to "Examine Patient" which already precedes "Collect Payment".

Conversely, eliminating a precedence link from α to β eliminates all direct and transitive precedence implied by this link. Assume that β precedes a third activity χ, thereby establishing transitive precedence from α to χ. If it is desired to maintain precedence from α to χ after removal of precedence link from α to β, a new precedence relation between α and χ has to be established in the same manner as discussed above.

Activity precedence is a partial ordering relation, i.e. it does not allow cycles. To respect this invariant, any adaptation of activity precedence must not create cycles, i.e. must not yield a transitive precedence where an activity α precedes itself, which yields an invalid pair $P(\alpha_{fi}, \alpha_{cr})$.

7.3.2.4 Adapt Activity Lifecycle

Adapting the lifecycle condition of an activity involves adapting its creation and fulfillment conditions as well as its repetition. Guidance on these operations applies whether performed separately or as part of adding a new activity.

Creation: Recall from 4.3.1.1 an activity is instantiated at the point where a participant may start performing it. Adapting the creation condition of an activity can thus be used to restrict the local behavior of a participant to comply with some business policy. For example, creation condition of "Examine Patient" may be adapted to require that Patient information must have been verified, which obliges a Doctor to verify Patient information, with an MP, prior to examining a Patient, i.e. Cr(Examine Patient) $\rightarrow$ O Verified(Patient.MPID).

Adapting the creation condition also serves to specify information required for an activity to be instantiated. The creation condition has to be constructed to be true only when this information is available, which can be specified directly by adapting its creation condition or, alternatively, indirectly via precedence (section 7.3.2.3). The "Request Verification" activity needs for its instantiation the Patient MPID to be verified. The availability of an MPID may be specified directly in its creation condition: Cr(Request Verification) $\rightarrow$ Available(Patient.MPID), or indirectly by adding a precedence link from the "Obtain MPID" activity to the "Request Verification", as in Figure 7.3.

Fulfillment: The fulfillment condition of an activity α is constructed such that it is true if and only if α has completed its work, whether work is performed directly by α, its children, or through delegation to another participant. For instance, the fulfillment condition of the Doctor's "Verify Patient Info" is Available(Patient.MPID) $\wedge$ Verified(Patient.MPID), which is achieved via its children, "Obtain MPID" and "Request Verification", which in turn complete their work by delegating to other participants. Additional work that an activity must complete can thus be specified either by adapting its fulfillment condition directly or that of its children, existing or newly created via refinement (section 7.3.2.2).

Since a precedence relation relates the creation condition of an activity to its fulfillment, adaptation of the creation condition may affect the fulfillment condition. Invariably, if the creation condition of an activity α implies that a condition Φ held in the past, the fulfillment condition of α will also imply that Φ held in the past: (Cr(α) $\rightarrow$ O Φ) $\rightarrow$ (Fi(α)$\rightarrow$O Φ). Consider for example where α is "Examine Patient" and Φ is Verified(Patient.MPID): Since Patient MPID

verification is required prior to executing "Examine Patient", then after completion of "Examine Patient" it is guaranteed that Patient MPID must have already been verified.

Repetition: For a repeating activity to be fulfilled, multiple instantiations of the activity may be required in one instance of an interaction. Adapting an activity to be repeating allows a participant to re-attempt its fulfillment if one or more instances fail. For example, annotating the "Make Appointment" activity as being repeating allows a Patient to re-request an appointment.

Annotating an activity as repeating also requires propagating the change to its children, since an instance of each child is instantiated for each instantiation of the parent, as per (6-15). The specification of repetition may also need to propagate to preceding activities. Consider where a Doctor provides to an MP incorrect Patient information, not only will they need to re-attempt performing "Request Verification" but also may need to repeat its preceding "Obtain MPID".

7.4 Guiding Change Propagation between Views

A valid specification of an interaction is composed of a set of consistent views (section 4.2.3). To maintain consistency between the global and local views, it is necessary to propagate changes between them. For every potential adaptation operation performed on the global/local view we identify the impact on the local/global view. Where applicable, we illustrate the propagation of changes arising from adaptations to the global and local views of the medical interaction that were worked out in section 7.3

7.4.1 Change Propagation from a Local View to the Global View

7.4.1.1 Propagate Adaptation of Activities

Whereas removal of an activity straightforwardly entails removing any dependencies attached to the activity, addition of an activity may involve making some decisions. A newly added activity α to the local view of a participant P_1 may require delegation of responsibility to another participant. After identifying the participants P_2 to which the responsibility is to be delegated, delegation is represented by adding a new dependency D to the global view whose endpoints are

in the local views of P_1 and P_2. A dependency designates data and/or control flow between participants (section 5.2.1):

- **Data Flow**: Reliance of activity α on P_2 for providing data is represented by making α the depender in the dependency D and making an activity in the local view of P_2 the dependee. If α has to provide data to P_2 to fulfill the dependency D, then D is annotated as bidirectional The fulfillment condition of D is contingent upon availability of the data to be provided. "Obtain MPID", when added to the Doctor's local view, needs to delegate responsibility to a Patient for providing their MPID. This requires adding an "MPID" dependency, to the global view, to represent this delegation. Since "Obtain MPID" is the recipient of information, it is made the depender in the dependency (Figure 7.8). The fulfillment condition of the dependency is Available(Patient.MPID).

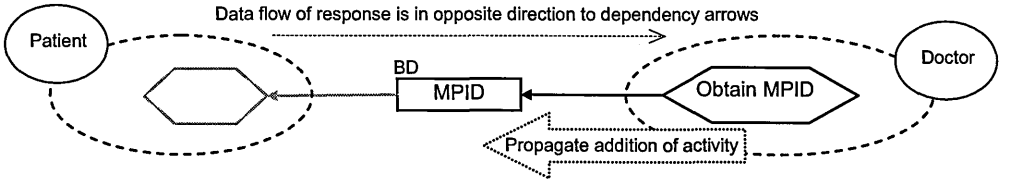


Figure 7.8 Propagate addition of activity by adding a data flow dependency

- **Control Flow**: A participant P_1 may require gaining control over interaction progression at certain points in execution. To have an activity α gain control from a participant P_2, P_2 is made to depend on α through a bidirectional dependency D. D is not fulfilled until α has been fulfilled, and fulfillment of D results in handing over control to P_2. Assume that adaptation to incorporate Patient MPID verification originated in the MP's local view. The MP adds a "Verify Eligibility" activity whose intent is to gain control from a Doctor at a point in the interaction prior to billing. A "Verification" dependency is thus added in which "Verify Eligibility" is the dependee and an activity in the Doctor's local view is the depender (Figure 7.9). Note that "Verification" designates both control and data flow.

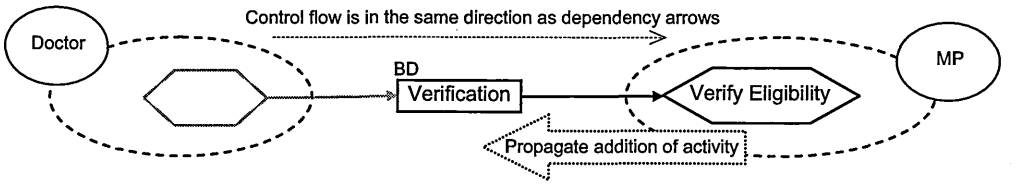


Figure 7.9 Propagate addition of activity by adding a control flow dependency

7.4.1.2 Propagate Adaptation of Activity Lifecycle

Whether done as part of adding a new activity or separately, adapting the lifecycle conditions and repetition of an activity α may need to propagate to dependencies attached to α.

Creation Condition: According to formula (6-2), the creation of a depender activity α in a dependency D must precede the creation of D, only if D is bidirectional. Adapting the creation condition of α must maintain the invariant $P(\alpha_{cr}, D_{cr})$. For example, if the creation condition of "Make Appointment" is adapted to require that "MPID must have been provided to Doctor", then the creation condition of the "Appointment" dependency may need to be adapted in order to require the same condition.

Fulfillment Condition: According to formula (6-1), a dependency D is not fulfilled unless the dependee activity β has been fulfilled. When the fulfillment condition of β is adapted, the fulfillment condition of any dependencies in which β is the dependee must be adapted to preserve the invariant $P(\beta_{fi}, D_{fi})$. This invariant is automatically preserved if Fi(D) is formulated in terms of O Fi(β) explicitly as in the specification of "Appointment" in Figure 4.5.

Repetition: Creation of an instance of a bidirectional dependency is always associated with an instance of its depender activity α, per formula (6-2). Each instantiation of α requires an instantiation of D, per formula (6-16), and adapting α to be repeating requires adapting each bidirectional dependency D in which α is the depender to be repeating as well.

7.4.1.3 Propagate Adaptation of Activity Precedence

Consider two activities α and β that are both connected to incoming and outgoing dependencies. Using formulas (6-1) and (6-2), we construct an event graph relating lifecycle events of the activities to the associated dependencies. Figure 7.10 shows the effect of adding a precedence link between the two activities on the lifecycle of associated dependencies.

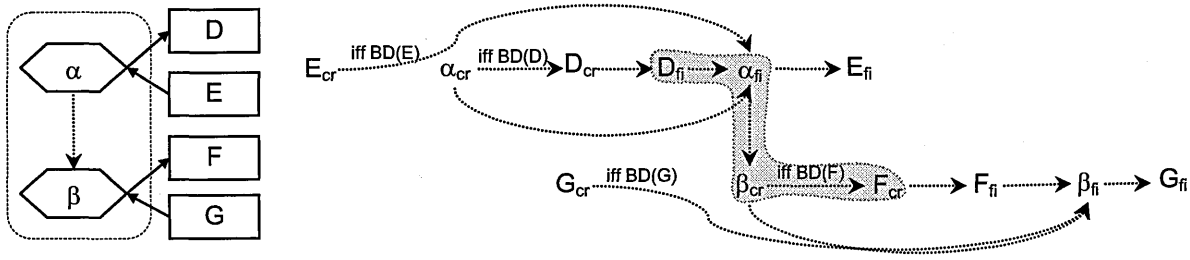


Figure 7.10 Impact of change in activity precedence on the global view

The event graph shows that adding a precedence link from α to β, i.e. adding $P(\alpha_{fi}, \beta_{cr})$, introduces transitive precedence between the fulfillment of the dependency D and the creation of the bidirectional dependency F, i.e. adds $P(D_{fi}, F_{cr})$, established through links inside the shaded area:

$$\mathbf{P(D_{fi}, \alpha_{fi}) \wedge P(\alpha_{fi}, \beta_{cr}) \wedge P(\beta_{cr}, F_{cr}) \rightarrow P(D_{fi}, F_{cr})}$$

Thus, in general, adding a precedence link from an activity α to an activity β is propagated to the global view by adding a precedence link from every dependency D in which α is the depender to every bidirectional dependency F in which β is the depender. For example, adding precedence link from "Request Verification" to "Collect Payment" in the Doctor's local view propagates to the global view by adding precedence between "Verification" and "Payment" dependencies as shown in Figure 7.11.

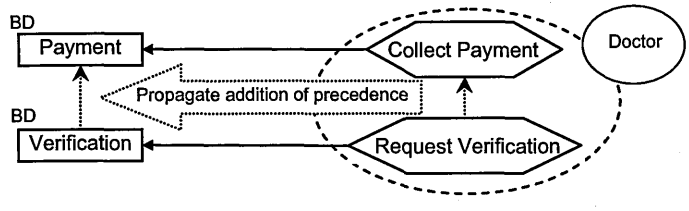


Figure 7.11 Propagate addition of precedence from "Request Verification" to "Collect Payment"

Additionally, since adding a precedence link from α to β constraints the creation of β, the creation condition of F may need to be adapted, according to section 7.4.1.2.

As a corollary, from formulas (6-7), (6-8), and (6-10), the aforementioned conclusions also apply for added precedence between any descendant of α and any descendant of β.

7.4.2 Change Propagation form the Global View to a Local View

7.4.2.1 Propagate Adaptation of Dependencies

Removing a dependency from the global view requires removing its attachments to the depender and dependee activities in local views. Adding a dependency requires propagating changes both to the local view of the depender role as well as the dependee role:

Dependee role: A dependee in a newly added dependency D is required to fulfill D. The dependee identifies an activity β which when fulfilled brings about the fulfillment condition of D. According to (6-1), β is chosen from the set of activities that at least satisfy: $Fi(D) \rightarrow O\ Fi(\beta)$. If this set is empty, the dependee creates a new activity whose purpose is to bring about the desired condition. In either case, the identified activity is made the dependee in D.

Depender role: A depender in a newly added dependency needs to detect its fulfillment. The depender either identifies an activity α whose fulfillment relies on the Fi(D) or creates one that does, and in both cases the identified activity is made the depender in D. From (6-1), α is chosen from that set of activities that at least satisfy $P(D_{fi}, \alpha_{fi})$. Additionally, from (6-2), if D is bidirectional, α must satisfy $P(\alpha_{cr}, D_{cr})$.

7.4.2.2 Propagate Adaptation of Dependency Precedence

Consider two dependencies D and E, using formulas (6-1) and (6-2), we construct an event graph relating lifecycle events of the dependencies to their depender and dependee activities. Figure 7.12 shows the effect of adding precedence link between two dependencies E and D on the associated activities in the two possible cases, where dependencies are in the same direction, case A, or opposing directions, case B.

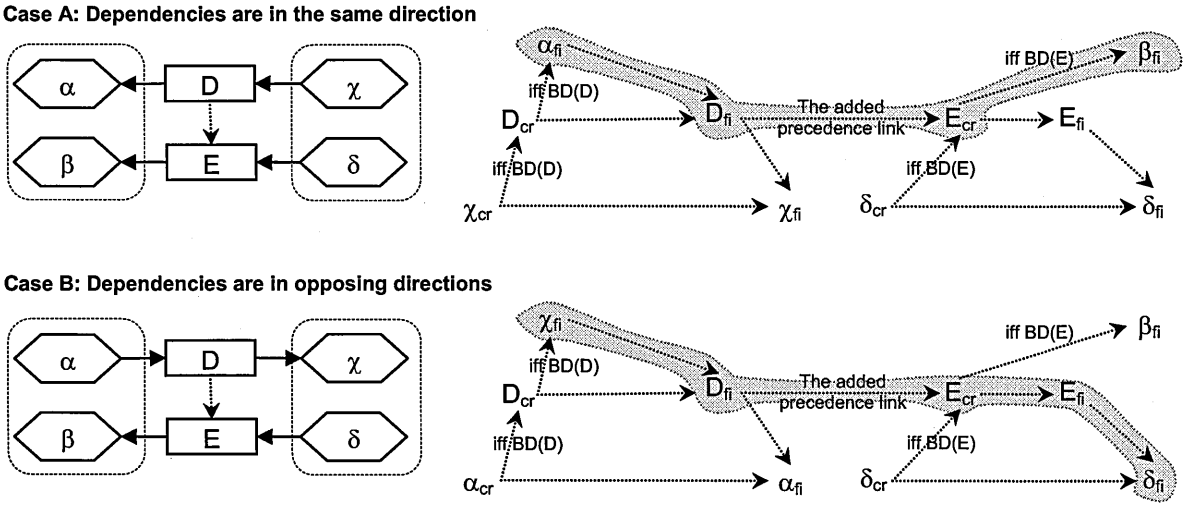


Figure 7.12 Impact of change in dependency precedence on local views

For case A, the event graph shows that adding a precedence link from D to E, i.e. adding $P(D_{fi}, E_{cr})$, introduces transitive precedence between the fulfillment of activity α, the dependee in D, and the fulfillment of the activity β, the dependee in E, only if E is bidirectional. Precedence pairs inside the shaded area establish precedence between α and β as follows:

$$\mathbf{P(\alpha_{fi}, D_{fi}) \wedge P(D_{fi}, E_{cr}) \wedge P(E_{cr}, \beta_{fi}) \rightarrow P(\alpha_{fi}, \beta_{fi})}$$

Thus, for any two dependencies D and E that have the same depender and dependee, where E is bidirectional, a new precedence link from D to E is propagated to the local view of the dependee in both dependencies by adding precedence from the fulfillment event of the dependee activity in D to fulfillment event of the dependee activity in E. For instance, adding a precedence link from the "Verification" dependency to the bidirectional "Payment" dependency propagates to the MP's view as a constraint that stipulates that the fulfillment of the "Pay Dues" activity precedes that of "Verify Patient Info". Precedence from the fulfillment of the latter to the fulfillment of the former can be achieved by making the fulfillment of the latter precede the creation of the former, i.e. by adding a precedence link between them (Figure 7.13).

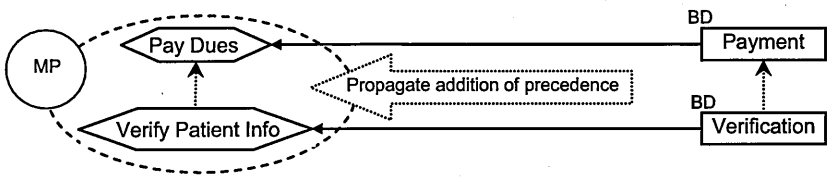


Figure 7.13 Propagating precedence between "Verification" and Payment to the local view of the MP

Case B in Figure 7.12 is tackled in a similar way where the shaded area in this case leads to the conclusion that: for any two dependencies D and E where the depender in D is the dependee in E and vice versa, adding a precedence link from D to E is propagated to the local view of the depender in E by adding precedence between the fulfillment of the dependee activity in D and the fulfillment of the depender activity in E.

7.4.2.3 Propagate Adaptation of Dependency Lifecycle

Adapting lifecycle conditions or repetition of a dependency D may require adapting lifecycle conditions and the repetition of depender activity α or dependee activity β as follows:

Creation: The creation event of D is related to lifecycle events of α and β only if D is bidirectional. Adapting the creation condition of D requires adapting the $Cr(\alpha)$ to maintain (6-2). For example, adapting the creation condition of the "Appointment" dependency to assert that "Patient's medical plan is active" requires the creation condition of "Make Appointment" in the Patient's local view to be adapted to assert the same condition, effectively limiting the Patient's ability to request an appointment.

Fulfillment: A depender activity is not fulfilled until its dependency has been fulfilled. Adapting the fulfillment condition of a dependency D may thus require adapting the fulfillment condition of α to maintain the invariant in (6-1). This invariant is automatically preserved if $Fi(\alpha)$ is formulated in terms of Fi(D) explicitly as in the specification of "Make Appointment" in Figure 4.5

Dismissibility: Adapting the dismissibility condition of a dependency may require adapting its creation condition to maintain their mutual exclusivity as per formula (6-5). Adapting the

creation condition of a bidirectional dependency requires adapting the creation condition of the depender activity α as described above.

Repetition: Fulfillment of an instance of D requires an instance of β, thus annotating D as being repeating requires annotating β as being repeating as well, per (6-17). For example, annotating the "Appointment" dependency as being repeating propagates to the Doctor's local view by annotating "Schedule Appointment" as repeating. Additionally, each instantiation of a bidirectional dependency corresponds to an instantiation of the depender activity. Annotating a bidirectional dependency as being repeating thus propagates to the depender activity α, as per (6-16). Thus, specification of repetition for the "Appointment" dependency also propagates to the "Make Appointment" activity in the Patient's local view.

7.5 Process for Adapting Interaction Specification

Sections 7.3 and 7.4 addressed the first four requirements for an adaptation framework stated in section 7.1, namely: guiding operations for adapting the global and local views and operations for change propagation between them. We weave these guidance techniques together into a process flow that achieves the fifth requirement; facilitating agreement between stakeholders on an interaction specification. The flow of the process is depicted in Figure 7.14, which is unsurprisingly consistent with Figure 7.2.

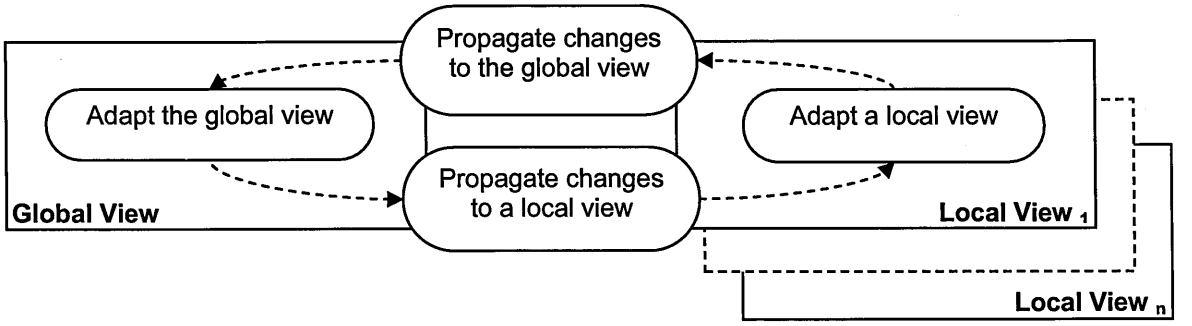


Figure 7.14 Process flow for adapting choreographed interaction specification

The process is initiated when an emergent requirement is identified, which requires adapting either the global view or one of the local views. Adapting their local view, a participant enumerates alternatives for adapting the view using guidance provided in section 7.3.2. As discussed in section 3.4.3, the participant evaluates alternatives from their point of view using their preferred method, e.g. model checking as introduced in section 4.3. An alternative is chosen, applied to the local view, and the impact of the change is propagated to the global view using techniques described in section 7.4.1. Since the global view is not owned by any participant, changes to the global view require approval from the global observer. Assuming the global observer finds the required changes reasonable, the global view is adapted using techniques described in section 7.3.1. Conversely, changes are propagated from the global view to affected local views, using techniques described in section 7.4.2, thereby keeping all views consistent. Allowing each stakeholder to reason about changes from their viewpoint enables collaboration between them towards obtaining a set of consistent views, i.e. reaching an agreement on an adapted interaction specification.

Bidirectional change propagation implies that the process is iterative. In iterations where a change is made, impact of the change is identified and propagated to affected views, thereby driving changes to be made in future iterations. The process terminates successfully if no more changes are required, i.e. all views are consistent indicating that an agreement has been reached. However, the process does not guarantee that an agreement will be reached eventually. The global observer may deem a requested change to the global view unfair and hence disallow it. At the same time a participant may be unable to find an acceptable alternative for adapting their local view, e.g. to achieve consistency with the global view, at which point the process fails. Such a conflict between participant views may be resolvable by seeking an alternative assignment of goal fulfillment responsibility at higher levels of abstraction (Bryl et al., 2006).

7.6 Chapter Summary

This chapter presented the third of our contributions: a framework for guiding the adaptation of interaction specifications. We started by stating the requirements that our framework should fulfill, which are guiding adaptation of global and local views, propagating changes between them, and forging an agreement between participants. We presented a metamodel for interaction specification based on a Tropos CLG diagram. In the course of adapting an interaction specification, validity is guaranteed via compliance with the metamodel and semantic invariants explicated in chapter 6. We detailed a catalogue of operations for adaptation of interaction specification views and change propagation between them. For every operation, we provided detailed guidance on exploration of alternatives and application of the operation in a way that ensures that a valid interaction specification is produced. We proposed an adaptation process that weaves together our guidance techniques to drive the forging of an agreement between participants and the production of a consistent set of interaction specification views. The next chapter demonstrates the utility of our adaptation process by applying it to a variety of case studies.

Chapter 8. Validation and Evaluation

This chapter validates and evaluates our adaptation framework and the ACDL generation technique. We evaluate our adaptation framework using two case studies. The first case study builds on the medical example introduced in Chapter 2 and revisited in Chapter 7. Resorting to a constructed case study facilitates the demonstration of our contributions in isolation from noise associated with a real world case study. The medical example is small enough to be comprehended with little effort and yet it allows for an illustrative application of our approach. The second case study brings together a well-studied example from the literature and real world requirements for vehicle accident insurance and repair. Tackling this example serves a dual purpose of benefiting from prior analysis that appeared in the literature and at the same time evaluating our approach in a real-world setting. Throughout the two case studies, we apply guidance provided in the catalogue of operations for adaptation and change propagation from chapter 7. Except for occasionally making justifiable assumptions about stakeholder intentions, we argue that we apply the operations in a systematic manner repeatable by an architect who studies this thesis. We demonstrate how our framework guides stakeholders through the process of adapting Tropos models that culminates in obtaining a messaging protocol using our automated tool. We reflect on the outcome pointing out strengths of our approach and limitations we have come across. Finally, we provide a comparative evaluation of our adaptation framework based on criteria extracted from the interaction specification and adaptation literature.

8.1 Validating Our Approach Using a Constructed Example

To demonstrate the utility of our adaptation framework we apply it to the running medical example. We identify two new requirements and apply our adaptation guidance to incorporate each of these requirements into the medical interaction specification. Once an adapted interaction CLG has been obtained, we use our tool to generate a messaging protocol from the adapted CLG. We apply the adaptation process in the way described in section 7.5. When adapting a local view, we put ourselves in the shoes of a local participant and take decisions in favor of their

business needs, making justifiable assumptions where necessary. We apply guidance pertinent to the local view adaptation operation being applied from section 7.3.2. Having made a change to a local view, we propagate it to the global view using guidance on change propagation from section 7.4.1. We put ourselves in the shoes of the global observer and judge the fairness of a change, then we apply the change using guidance for the relevant operation from section 7.3.1. Having made changes to the global view, we propagate changes to all affected local views using guidance from section 7.4.2. We keep iterating until no further changes need propagation. The starting point for both adaptations is the CLG model of Figure 4.4, which is repeated in Figure 8.1, after adding dependency annotations and replacing the unidirectional "Invoice" and "Payment" dependencies with a single bidirectional "Payment" dependency, as discussed in section 5.2.1.

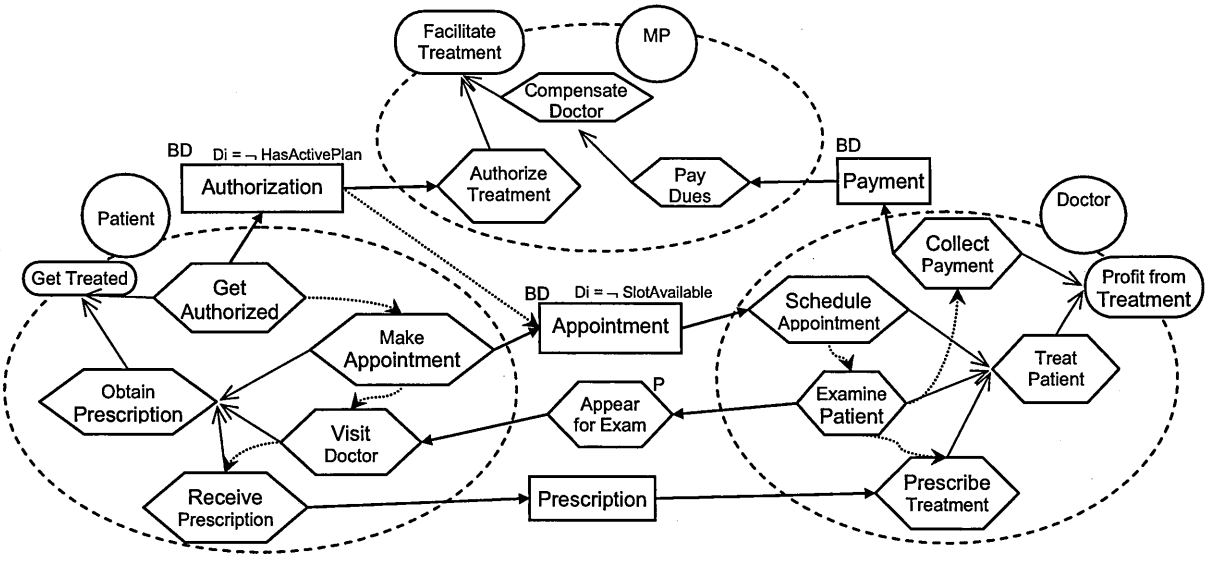


Figure 8.1 Starting-point CLG of the medical interaction adaptation

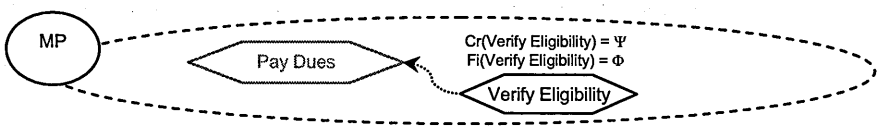
8.1.1 First Adaptation: Verifying Patient's Eligibility for Treatment

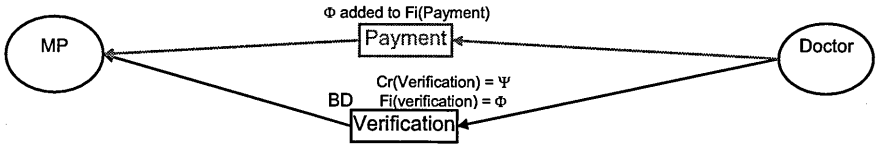
The MP discovers the following business need: prior to paying a Doctor the MP has to verify that treatment was performed on an eligible patient, i.e. a patient that has valid coverage and an applicable authorization. We apply our adaptation process to incorporate this requirement into the CLG model of Figure 8.1, and then we generate the adapted ACDL.

8.1.1.1 Applying the Adaptation Process

Starting from the emergent requirement, we detail the steps of applying our adaptation process. The steps are numbered for ease of reference. We also annotate each step instances with the type of the step being applied: Adapt Local view as AL, Propagate to Global view as PG, Adapt Global view as AG, and Propagate to Local view as PL. Additionally, we reference the section from Chapter 7 for the operation being applied in each step. After a set of logically related adaptation steps we present a CLG fragment summarizing elements added in these steps as well as elements existing in the original CLG that were modified (shown in grey). The process proceeds as follows:

1	An MP needs to verify eligibility of a Patient for treatment, and ensure this verification takes place prior to reimbursing a Doctor. To fulfill this two-part need (Φ denotes the condition that "Patient eligibility has been verified by MP" and Ψ denotes that "Patient Medical Plan ID (MPID) is available" to a certain activity) : (AL 7.3.2.1 Add Activity) The MP adds an activity for verifying Patient eligibility, "Verify Eligibility", to their local view. This new activity is responsible for verifying the eligibility of a Patient for coverage, given her MPID. For the new activity to verify eligibility it requires that the Patient's MPID is available. Hence, its creation condition is Ψ, and since after it completes Patient MPID has been verified, its fulfillment condition is Φ. (AL 7.3.2.3 Add Activity Precedence) To ensure that no payment request from a Doctor is processed until Patient eligibility verification is complete, i.e. until Φ is true, an MP needs to constraint the execution order of their activities. Since the activity that brings about Φ is "Verify Eligibility" and the activity that processes payments is "Pay Dues", the MP adds a precedence link from the former to the latter.

	
2	Two changes were made to the MP's local view, each of which has an impact that needs to propagate to the global view as follows: (PG 7.4.1.1 Propagate Added Activity) The newly added "Verify Eligibility" activity needs to satisfy Ψ for its creation, i.e. it needs the MPID of the Patient in question to be provided (by the Doctor receiving a Patient). Additionally, it needs to gain control of the progression of the interaction (from the Doctor) to perform the eligibility check prior to the Doctor requesting a payment. These two needs imply that the MP requires the doctor to provide Patient MPID and relinquish control of interaction progression, which can be achieved via a new dependency that serves both the data and control flow purposes. Thus, the MP suggests adding a "Verification" dependency to the global view where "Verify Eligibility" is dependee and choice of depender activity is delegated to Doctor. Its fulfillment relies on the completion of "Verify Eligibility" and the dependency is dismissed if the activity fails to verify Patient MPID. (PG 7.4.1.3 Propagate Added Activity Precedence) Adding a precedence link from "Verify Eligibility" to "Pay Dues" constraints the fulfillment of the latter such that "Pay Dues" may not be fulfilled until Patient MPID has been verified, i.e. until Φ becomes true. Since the fulfillment of "Pay Dues" is necessary for the fulfillment of any dependency in which it is the dependee, Φ is suggested by the MP a necessary pre-condition for the "Payment" dependency to be fulfilled.
3	Two changes to the global view were suggested by the MP: adding a "Verification" dependency and adding Φ as a pre-condition on the fulfillment of "Payment". (AG 7.3.1.1 Add Dependency) The suggested addition of a "Verification" dependency is found to be reasonable and is approved by the regulatory agency. The new dependency is

	added to the global view where its creation condition is Ψ and fulfillment condition is Φ. (AG 7.3.1.3 Adapt Dependency Lifecycle) Also, the suggested change to make Φ a pre-condition on the fulfillment of "Payment" is applied by making Φ a necessary fulfillment condition of the dependency. 
4	The addition of "Verification" dependency as well as the change in the fulfillment condition of "Payment" dependency need to be propagate to the local view of the depender in both dependencies, the Doctor. (PL 7.4.2.1 Propagate Added Dependency) Since it is in their interest to ensure that they get paid, Doctor deems the the added "Verification" dependency reasonable and accepts the responsibility. (PL 7.4.2.3 Propagate Adaptation of Dependency Lifecycle) Since "Collect Payment" is not fulfilled until "Payment" has been fulfilled, the added fulfillment condition of the dependency is also added to the fulfillment condition of the activity. That is, Φ is made a necessary fulfillment condition of the "Collect Payment" activity.
5	The Doctor explores alternatives for applying the two changes propagated from the global view: first, choosing a depender activity for the "Verification" dependency and specifying its lifecycle conditions, and second, enforcing the newly added condition Φ on the fulfillment of "Collect Payment". Propagating the first change triggers further changes and requires several steps: (AL 7.3.2.1 Add Activity) Since no existing activity in the Doctor's local view bears the responsibility of requesting verification of Patient MPID, a new "Request Verification"

activity is added and made the depender in "Verification".

(AL 7.3.2.4 Adapt Activity Lifecycle) Doctor specifies lifecycle condition for the newly added activity. By being the depender in "Verification", the activity is fulfilled when the Patient MPID has been verified, thus its fulfillment condition is Φ. To request verification, a Doctor must have the MPID of the Patient, hence Ψ is made a necessary creation condition of the activity consistent with the creation condition of the dependency.

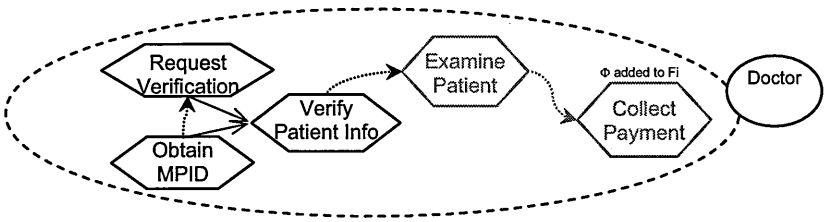
(AL 7.3.2.4 Adapt Activity Lifecycle) Having added Ψ as a creation condition of "Request Verification", Doctor attempts to identify an activity in their local view that can bring this condition about.

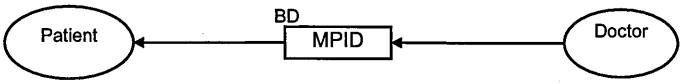
(AL 7.3.2.1 Add Activity) No such activity is found, and so Doctor adds an "Obtain MPID" to make the MPID available and bring about Ψ. "Obtain MPID" is made to precede "Request Verification" to enforce the creation condition of the latter. Logically, the creation condition of "Obtain MPID" stipulates that it is created only when a Patient has requested an appointment.

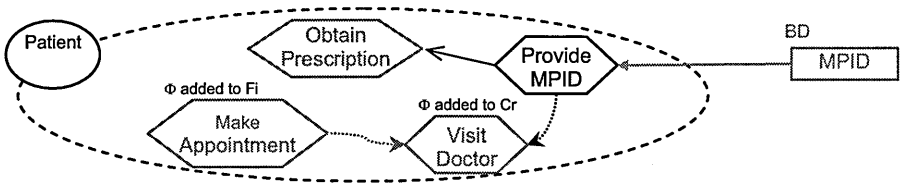
(AL 7.3.2.2 Adapt Refinement) For modularity, since both "Request Verification" and "Obtain MPID" perform work pertaining to Patient information verification, they are grouped together as children of a new activity "Verify Patient Info". The AND-refinement of the activities implies that the fulfillment condition of the newly added parent is the conjunction of fulfillment conditions of its children, that is: $\Phi \wedge \Psi$. The creation condition of the parent is the same as its children, which is that a patient has requested an appointment.

To propagate the second change to their local view, Doctor explores alternatives for enforcing the newly added fulfillment condition Φ on "Collect Payment".

(AL 7.3.2.4 Adapt Activity Lifecycle) Fulfillment of "Verify Patient Info" brings about $\Phi \wedge \Psi$, i.e. it brings about Φ. Making "Verify Patient Info" precede "Collect Payment"

	would enforce that the latter is not fulfilled until Φ becomes true. However, through model checking or inspection, Doctor finds that this alternative allows a scenario where a Patient is found to be not eligible, *after* being examined. This alternative is thus deemed unsatisfactory as it allows an undesirable scenario where Doctor is denied payment. (AL 7.3.2.3 Adapt Precedence) Instead of direct precedence, Doctor explores ways for establishing transitive precedence between "Verify Patient Info" and "Collect Payment". Doctor decides to add precedence from "Verify Patient Info" to "Examine Patient", which already precedes "Collect Payment" to establish transitive precedence. 
6	(PG 7.4.1.1 Propagate Add Activity) The newly added "Obtain MPID" activity requires Patient to provide information, thus the effect of adding the activity needs to propagate to the global view. Doctor suggests adding an "MPID" dependency to the global view to capture their reliance on Patient for providing her MPID. Since data flows from Patient to Doctor, the Patient is made the dependee and the Doctor the depender. Doctor suggests making the dependency bidirectional so that they control the time at which to request its fulfillment. Thus, they suggest that an instance of the dependency is only created when the depender activity "Obtain MPID" has been created.
7	(AG 7.3.1.1 Add Dependency) The regulatory agency deems it appropriate for a Doctor to request a Patient's MPID and thus allows adding the new dependency. A bidirectional "MPID" dependency is added to the global view. The creation condition of the dependency carries over from its depender activity "Obtain MPID", which is that Patient has requested an appointment. Fulfillment condition of the dependency is that MPID has been provided, i.e. Ψ.

	
8	The added MPID dependency needs to propagate from the global view to the Patient's local view. (PL 7.4.2.1 Propagate Add Dependency) Patient accepts the newly added "MPID" dependency and thereby takes on the responsibility of providing her MPID to Doctor upon request.
9	Patient adapts their local view to reflect the new responsibility for providing an "MPID". (AL 7.3.2.1 Add Activity) Patient adds a new "Provide MPID" activity and makes it the dependee in the "MPID" dependency. Its creation condition is that the "MPID" dependency has been created and its fulfillment is that the Patient has provided their MPID. (AL 7.3.2.2 Adapt Refinement) Since "Obtain Prescription" is now contingent upon providing her MPID, and since "Provide MPID" is also a necessary part of activities leading to the obtainment of a prescription, "Provide MPID" is made a child of "Obtain Prescription". Patient determines that the adapted model allows a Doctor to request her MPID after "Visit Doctor" has been fulfilled, i.e. the Patient may be required to carry a physical form of the MPID to the Doctor's office. In this scenario, if they forget to carry the proof they may get denied examination even after going through trouble of visiting Doctor's office, and hence the Patient finds this scenario undesirable. (AL 7.3.2.3 Adapt Activity Precedence) To guarantee that they can only be asked to provide their MPID before visiting the Doctor, Patient decides to make "Provide MPID" precede "Visit Doctor". However, Patient finds this option unsatisfactory as well since Doctor may obtain MPID from Patient before they visit, but only ask MP to verify it after

	Patient has visited their office. (AL 7.3.2.4 Adapt Activity Lifecycle) To guarantee that they do not visit a Doctor until their MPID has been verified, Patient decides to add Φ as a necessary condition on the creation of "Visit Doctor". That is, they will not begin executing the "Visit Doctor" task until their MPID has been verified. Since "Make Appointment" already precedes "Visit Doctor", adding a fulfillment condition to the former carries over to the creation condition of the latter. Thus, they decide to add Φ as a necessary fulfillment condition of "Make Appointment" thereby stipulating that verification of their MPID is now part of setting up an appointment. 
11	(PG 7.4.1.2 Propagate Adaptation of Activity Lifecycle) Since "Make Appointment" is the depender in the "Appointment" dependency, the change in the fulfillment condition of the activity needs to be propagated to the global model.
12	(AG 7.3.1.3 Adapt Dependency Lifecycle) The regulatory agency deems the Patient's requirement, for completion of verification prior to finalizing an appointment, reasonable and agrees to it. The fulfillment condition of "Make Appointment" needs to brought about by the "Appointment" dependency, and hence Φ is added to the fulfillment condition of "Appointment".
13	(PL 7.4.2.3 Propagate Adaptation of Dependency Lifecycle) Doctor accepts the new responsibility implied by the new fulfillment condition of "Appointment", in which they are the dependee, and propagates the change to their local view.

14	(AL 7.3.2.4 Adapt Activity Lifecycle) Since "Schedule Appointment" is already responsible for fulfilling the "Appointment" dependency, Doctor explores options for having the activity bring about the new necessary condition, Φ, required for fulfilling the dependency. Since "Verify Patient Info" is already responsible for bringing about Φ, and since MPID verification is now part of appointment scheduling, Doctor decides to make "Verify Patient Info" a child of "Schedule Appointment". The fulfillment condition of the former now becomes a necessary fulfillment condition of its parent.
15	At this point none of the participants requires further changes. All participants agree on the adapted model.

Given that original model in Figure 8.1, Figure 8.2 summarizes model elements that were added or changed. We omit original model elements that have not been changed but include ones that have been linked to, and gray them out.

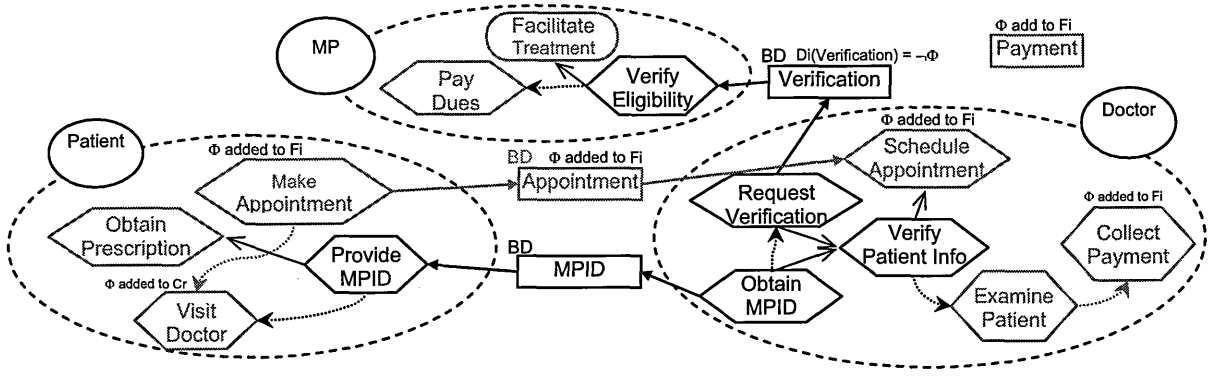


Figure 8.2 Summary of first set of adaptations made to the medical example

8.1.1.2 Generating ACDL Protocol

We applied our automated tool to the adapted model to obtain the ACDL messaging protocol specification in Figure 8.3. Note how this protocol specification ensures the patient will never obtain an appointment unless they provide proof of eligibility.

Even with such a tailored example, the automation provided by the tool has proven useful in avoiding specification errors. Prior to finishing the tool implementation, we had made an honest specification mistake in an earlier incarnation of the medical example (Mahfouz et al., 2009) where we hand-constructed a messaging specification that was supposed to be consistent with the corresponding requirements model. A mistake was later revealed after running the tool on the requirements model, long after the work has been published. Our tool demonstrated that the hand-constructed messaging specification was indeed inconsistent with the requirements model.

```
Sequence
   Send AuthorizationRequest From Patient To MP
   If (HasActivePlan)
      Send AuthorizationResponse From MP To Patient
   Else
      Sequence
         Send AuthorizationRejected From MP To Patient
         Fail 'NOT HasActivePlan'
   Send AppointmentRequest From Patient To Doctor
   Send MpidRequest From Doctor To Patient
   Send MpidResponse From Patient To Doctor
   EligibilityProvided = true
   Send VerificationRequest From Doctor To MP
   If (ValidMPID)
      Send VerificationResponse From MP To Doctor
   Else
      Sequence
         Send VerificationRejected From MP To Doctor
         Fail 'NOT ValidMPID'
   MpidVerified = true
   EligibilityVerified = true
   If (SlotAvailable)
      Send AppointmentResponse From Doctor To Patient
   Else
      Sequence
         Send AppointmentRejected From Doctor To Patient
         Fail 'NOT SlotAvailable'
   Parallel
      Sequence
         Send PaymentRequest From Doctor To MP
         Send PaymentResponse From MP To Doctor
      Send PrescriptionResponse From Doctor To Patient
```

Figure 8.3 Generated ACDL for the first adaptation of the medical example

8.1.1.3 Discussion

Whereas physical activities are not directly represented in the resulting messaging specification, performing the adaptation to the requirements models allowed us to take into account physical activities both as:

a. Constraints: The need to verify eligibility before the Patient visits the Doctor played a role in deciding the order of messaging activities.

b. Alternative realization of dependencies: Carrying physical proof of eligibility to the Doctor's office was considered as an alternative to providing eligibility information via electronic messaging.

It can be argued that the Doctor could have realized the need to make "Verify Patient Info" a child of "Schedule Appointment" in step 5 rather than later. We deliberately delayed establishing this parent-child relation to further demonstrate the collaborative nature of our process. Later in the process, in step 10, Patient incorporates an ordering constraint to guard their interests, which is then propagated back to Doctor's local model where the omitted relation gets established. After Doctor added the missing relation, the produced model satisfies constraints imposed by both local models, the Doctor's and the Patient's. Even though the longer path produced redundant constraints, e.g. the precedence between "Provide MPID" and "Visit Doctor", our automated tool manages to generate an ACDL specification while eliminating redundant constraints.

There is no guarantee that the process will yield a single canonical adaptation. For example, Doctor could have added precedence from "Verify Patient Info" to "Schedule Appointment" and the resulting adapted model would have required Patient to provide her MPID before even requesting an appointment. However, if a process concludes successfully, i.e. all participants agree to an adapted global model and all changes have been propagated, our adaptation guidance guarantees that the resulting adaptation is a valid one that satisfies the business needs of all participants. We do not provide means for finding a "shortest path" to adaptation or means for obtaining a minimal messaging specification.

8.1.2 Second Adaptation: Limiting Outstanding Balance

The second adaptation aims to improve the cash flow of a Doctor. Since an MP is not required to provide immediate reimbursement, a Doctor needs protection against a delinquent MP. To mitigate this risk, the health department, i.e. the regulatory agency, requires maintaining the outstanding balance within a certain credit limit for every Doctor-MP pair. The outstanding

balance is the total of all payments that have been requested but not yet paid. We apply our adaptation process to incorporate the new constraint into the CLG model of Figure 8.1.

8.1.2.1 Applying the Adaptation Process

The outstanding balance of an MP increases whenever a Doctor requests a payment from an MP, hence the health department requires as a pre-condition on requesting a payment that condition Θ holds, where Θ represents "credit limit has not been exceeded". The process proceeds as follows:

1	The health department suggests an adaptation to the global model so as to incorporate the pre-condition on requesting a payment. (AG 7.3.1.3 Adapt Dependency Lifecycle) Since requesting a payment is tied to the creation of a "Payment" dependency, the pre-condition can be enforced by adapting the creation condition of the dependency. Hence, Θ is added as a necessary creation condition for the "Payment" dependency.
2	Since Doctor is the depender in the bidirectional "Payment" dependency, a Doctor is responsible for its instantiation. Thus, adaptation to its creation condition needs to propagate to their local model. (PL 7.4.2.3 Propagate Adaptation of Dependency Lifecycle) Doctor identifies the depender activity in the "Payment" dependency. An instance of "Collect Payment" is responsible for creating an instance of "Payment". Thus, Θ needs to be added as a necessary pre-condition on the creation of "Collect Payment".
3	Doctor explores options for adapting their local view to incorporate Θ as a necessary pre-condition for "Collect Payment". (AL 7.3.2.4 Adapt Activity Lifecycle) Doctor finds the option of adding Θ as a necessary creation condition on "Collect Payment" is undesirable. This adaptation allows a state

	where a Doctor finds out they cannot request a payment from an MP except AFTER having provided treatment to Patient, thereby potentially wasting their effort if the MP fails to reimburse them. Doctor considers other options for making Θ a pre-condition on "Collect Payment" including adding Θ as a necessary fulfillment condition on activities that precede it. Doctor finds a satisfactory alternative which is to add Θ as a fulfillment condition of "Schedule Appointment", whose fulfillment transitively precedes the creation of "Collect Payment". (AL 7.3.2.1 Add Activity) Doctor's local view does not contain any activity that can bring about the condition Θ. Doctor adds a "Check Limit" activity whose purpose is to assert that Θ true, and thus its fulfillment condition is Θ. (AL 7.3.2.2 Adapt Refinement) To enforce that Θ is part of the fulfillment condition of "Schedule Appointment", "Check Limit" is added as its child thereby making the fulfillment condition of the latter, i.e. Θ, a necessary part of the former.
4	(PG 7.4.1.2 Propagate Adaptation of Activity Lifecycle) Since "Schedule Appointment" is the dependee in the "Appointment" dependency, changes made to its lifecycle need to propagate to the global view. Since the activity is responsible for fulfilling the dependency, changes in its fulfillment thus propagate to the "Appointment" dependency. Doctor also suggests dismissing the "Appointment" dependency if the credit limit of the Patient's MP has been exceeded, i.e. if "Check Limit" fails and Θ does not hold.
5	(AG 7.3.1.3 Adapt Dependency Lifecycle) Θ is added as a necessary fulfillment condition of the "Appointment" dependency. Also, the health department agrees to make the dependency dismissible if the credit limit of the MP in question has been exceeded.
6	(PL 7.4.2.3 Propagate Adaptation of Dependency Lifecycle) Change in fulfillment and

	dismissibility conditions of the "Appointment" dependency need to propagate to its depender activity, "Make Appointment".
7	(AL 7.3.2.4 Adapt Activity Lifecycle) Patient may object to the new dismissibility condition since maintaining the balance within the credit limit is the MP's responsibility, and not theirs. If Patient objects, the process terminates in failure. We will assume that Patient does not object so that we illustrate the process further. The Patient propagates the change in "Appointment" dismissibility condition to "Make Appointment" activity. Patient explores alternatives to guarantee that they can re-attempt to obtain an appointment. They annotate "Make Appointment" as being repeating.
8	Changes to the Patient's local view propagate to the global view. (PG 7.4.1.2 Propagate Adaptation of Repetition) Adaptation of repetition of "Make Appointment" propagates to the dependency in which it is the depender.
9	(AG 7.3.1.3 Adapt Dependency Repetition) The health department agrees to annotating the "Appointment" dependency as being repeating to allow a Patient to re-request an appointment.
10	Change in "Appointment" repetition needs to propagate to the local view of the Doctor. (PL 7.4.2.3 Propagate Adaptation of Dependency Repetition) Repetition of the "Appointment" dependency propagates to the dependee activity "Schedule Appointment" which is marked as repeating.
11	(AL 7.3.2.4 Adapt Activity Repetition) Doctor annotates the "Schedule Appointment" activity as being repeating.
	None of the participants requires further changes, they agree to the adapted model.

8.1.2.2 Generating ACDL Protocol

A summary of the adaptations is shown in Figure 8.4 and the generated ACDL generated from the adapted model is shown in Figure 8.5. The response to an appointment request is now guarded by the condition on the credit limit. If the check fails the appointment is rejected and another iteration of the "While" loop starts. Note that, as specified, the loop only terminates if the Patient obtains an appointment. Practically, a Patient, after trying a number of times, may give up and abandon the interaction.

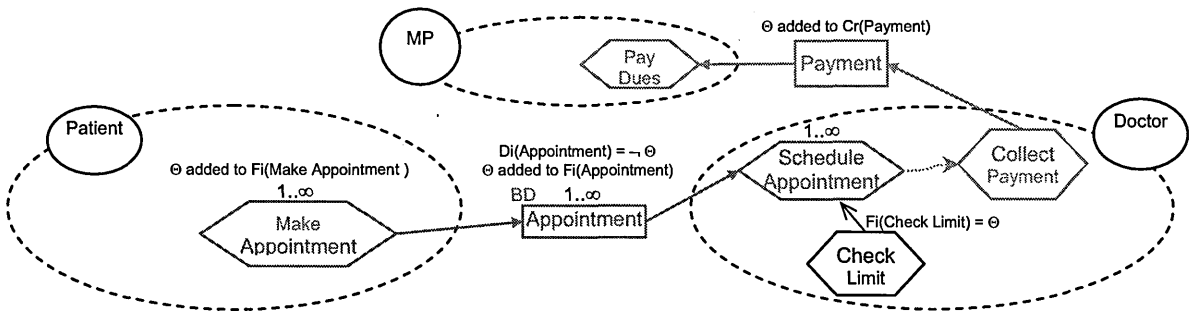


Figure 8.4 Summary of second set of adaptations made to the medical example

```
Sequence
   Send AuthorizationRequest From Patient To MP
   If (HasActivePlan)
      Send AuthorizationResponse From MP To Patient
   Else
      Sequence
         Send AuthorizationRejected From MP To Patient
         Fail 'NOT HasActivePlan'
   While (NOT AppointmentObtained)
      Sequence
         Send AppointmentRequest From Patient To Doctor
         If (SlotAvailable_AND_WithinCreditLimit)
            Sequence
               Send AppointmentResponse From Doctor To Patient
               AppointmentObtained = true
               Parallel
                  Sequence
                     Send PaymentRequest From Doctor To MP
                     Send PaymentResponse From MP To Doctor
                  Send PrescriptionResponse From Doctor To Patient
         Else
             Send AppointmentRejected From Doctor To Patient
```

Figure 8.5 ACDL incorporating the credit-limit check adaptation

8.1.2.3 Discussion

A viable adaptation alternative, which we had otherwise identified was not discovered by our process. An MP who believes they have exceeded their credit limit with a Doctor could stop issuing authorizations to patients, i.e. dismiss authorizations. The reason this alternative was not discovered is that the emergent requirements was expressed from the global point of view. Had we formulated the requirement from the MP's local point of view, for instance as "authorizations may only be issued when credit limit has not been exceeded", our process would have found the alternative that involves dismissing authorizations. However, a Doctor has to protect their interests against an MP who fails to reject an authorization. A Doctor should be allowed to dismiss an appointment when the credit limit has been exceeded.

Even though conditions under which an appointment may be dismissed are not relevant to a Patient, the multi-participant nature of a choreographed interaction caused Patient's interests to be affected by the relation between MP and Doctor. Had a Patient rejected the potential of dismissing appointment requests, the adaptation process may have ended in failure. In reality, the Patient's opinion may not have the same weight as that of the MP, so they could end up being forced to comply with the adapted specification. Our process assumes all participants have an equal say and hence does not account for such situations.

8.2 Validating Our Approach Using a Real-World Scenario

Government agencies overseeing the insurance business put forward rules to protect the interests of vehicle owners as well as all parties involved in vehicle insurance and repair. Interactions that take place after an accident between vehicle owner, insurer, and repairers are typically regulated. Our second validation experiment combines a "Vehicle Insurance" case study frequently occurring in the interaction specification literature with requirements drawn from publicly available real world documents that regulate the vehicle insurance business. We adapt the specification of the vehicle insurance interaction as it appears in the literature by incorporating requirements extracted from the real-world documents. First, we introduce the specification of vehicle insurance interaction as described in the literature, then analyze

documents that regulate vehicle insurance business in North America, adapt the literature specification to a variety of requirements stated in the real-world documents, and reflect on the results. We apply our ACDL derivation tool before and after adaptation to obtain, in an automated way, a messaging protocol consistent with the requirements models.

8.2.1 Requirements for Vehicle Repair from Literature – AGFIL Case

The original specification of the vehicle repair interaction first appeared as the "AGFIL case study" (Browne & Kellet, 1999) in the CrossFlow project (Grefen et al., 2000). The case study was later revisited by several authors (Xu, 2004; Desai et al., 2006; Orriëns & Yang, 2006; Desai et al., 2009). We present the original interaction specification then build on the analysis conducted later by Telang & Singh (2009) to create Tropos requirements models for the interaction. Where these Tropos models diverge from the original specification, we stick to the original. On the other hand, where these Tropos models are under-specified we make justifiable assumptions to fill the gaps. We refer to the literature case, both the original specification and the corresponding Tropos models, as the AGFIL case for short.

8.2.1.1 Original Specification of Vehicle Repair Scenario

AGFIL is an insurance company that covers the cost of vehicle damage repair incurred by policy holders. AGFIL provides claim reception and vehicle repair services to policy holders. Additionally, AGFIL needs to assess claims to protect itself against fraud. AGFIL uses its partners, Europ Assist (EA), Lee Consulting Services (CS), and various repairers, for executing these tasks. EA provides a help-line to policy holders for reporting a claim, and directs them to an approved repairer facility. Lee CS provides insurance adjustors, who perform damage assessment, and presents invoices to AGFIL on behalf of repairers. Several approved repairers provide repair services at their shops. AGFIL makes decisions on claim approvals, and provides payment to repairers. Figure 8.6 depicts the original AGFIL case reproduced from (Browne & Kellet, 1999) where arrows represent control and data flow between activities.

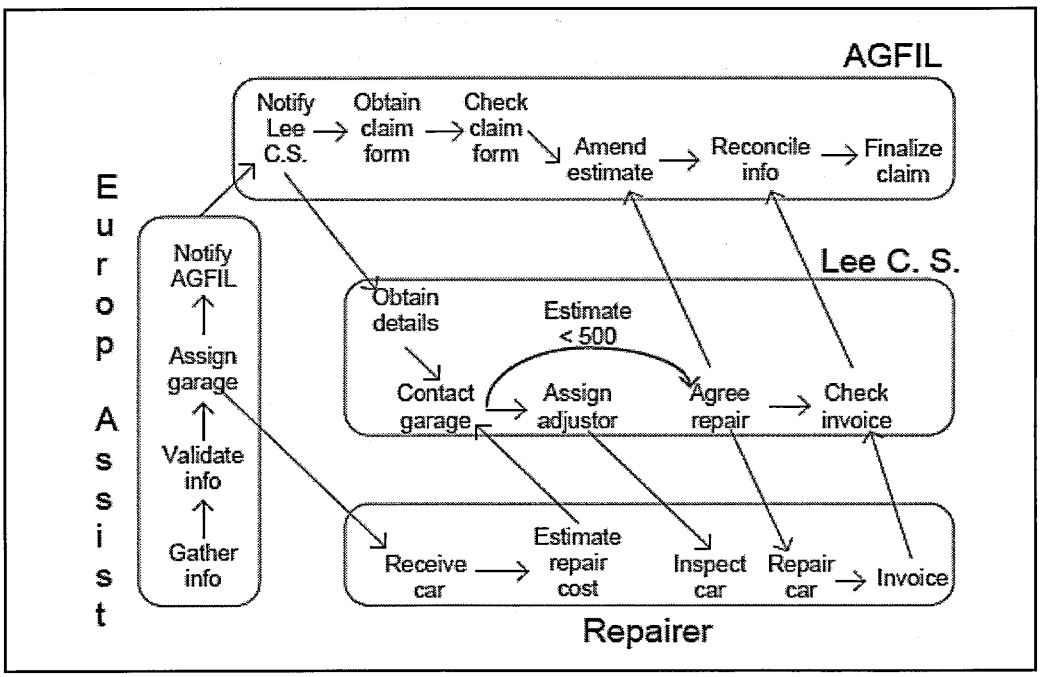


Figure 8.6 Original AGFIL flow specification.

8.2.1.2 Global View – High-level Role-Dependency Diagram

The AGFIL interaction involves five roles: Claimant, Insurer, Adjustor, Repairer, and Call Center. The original case study mostly ignored the Claimant role after reporting the accident, because the case study was focusing on the Insurer business process. To construct an RD diagram for the choreographed interaction, we include dependencies between Claimant and the other roles (Figure 8.7). The scope of the interaction is limited to the claim processing and vehicle repair. That is, signing up for insurance, collecting insurance premiums, paying adjustor's fees, etc. are outside the scope of the interaction, consistent with the original specification.

Analyzing dependency characteristics we find that among the total of fourteen dependencies three are physical by nature: "Release Vehicle" where Claimant hauls vehicle to Repairer, "Present Vehicle" where Repair presents vehicle to Adjustor for inspection, and "Repaired Vehicle" where Repairer releases back repaired vehicle to Claimant. We also chose physical fulfillment for "Accident Info" as it is common that Claimant reports an accident on-site via phone, consistent with the original specification. Otherwise, the rest of the dependencies are

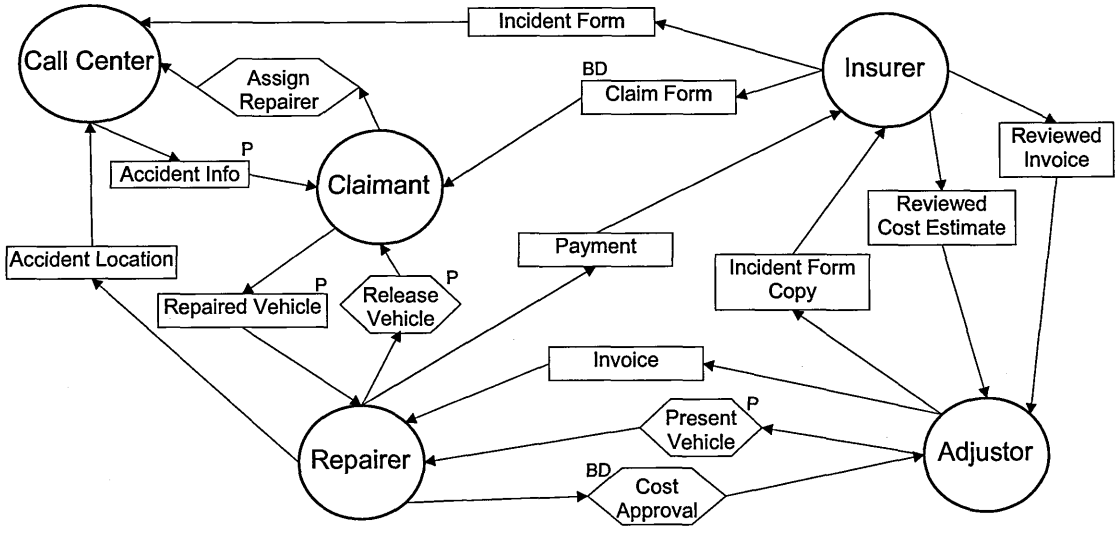


Figure 8.7 Global model of vehicle repair example.

fulfilled via electronic messaging. We also identified two bidirectional dependencies: "Claim Form", where Insurer requests Claimant to fill form, and "Cost Approval", where Repairer requests Adjustor to approve repair costs.

8.2.1.3 Local Views and Combined Local-Global Diagram

Having specified the global view of the AGFIL interaction, we construct the local view for each participant. We used the analysis of the AGFIL interaction presented by Telang & Singh (2009) as a starting point. This analysis concluded by providing participant local views including activities and their refinement. Additionally, the original AGFIL use case specification contains several data and control flow constraints on participants activities (Browne & Kellet, 1999). For example, as depicted using arrows in Figure 8.6, a Repairer receives a car, then estimates repair cost, then they contact an Adjustor, which in turn inspects the car and approves the cost, then they repair the car only when the estimate has been approved. We extracted such data and control flow requirements from the original specification and represented them using precedence links in the local views contributed by (Telang & Singh, 2009). Finally, using dependencies from the global model, we combined the local models to construct the CLG diagram in Figure 8.8. All precedence links in the figure are imposed by the original flow (Browne & Kellet, 1999), except for the link between the "Perform Repair" to the "Release Vehicle" activities which is logically necessary.

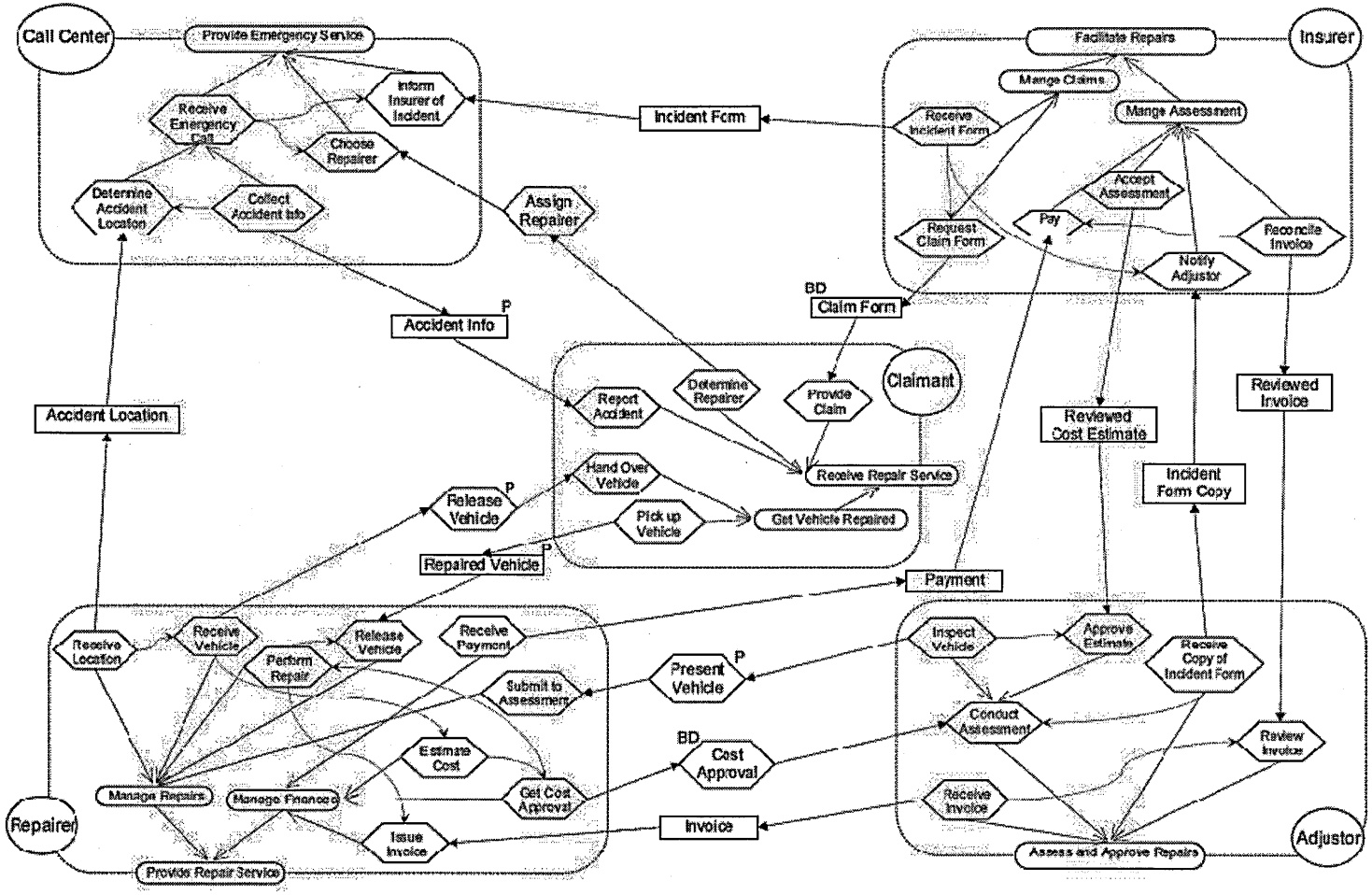


Figure 8.8 Combined local-global model for the vehicle repair example

8.2.1.4 Generating ACDL Protocol

Feeding the GLG to our tool we automatically obtain the messaging protocol in Figure 8.9. Two notes on how the generated protocol reflects the constraints embedded in the CLG:

- Parallelism: Call Center simultaneously notifies Insurer of an incident and informs Claimant of an assigned repairer. We wondered when we first saw the protocol, why wouldn't Call Center also simultaneously notify Repairer of accident location. The CLG explained the reason that "Determine Accident Location" is a child of "Receive Emergency Call" and hence its fulfillment precedes that of its parent. Had we constructed the model such that the two precedence links emanating from "Receive Emergency Call" are instead emanating from "Collect Accident Info", the higher level of parallelism would have been obtained.
- Synchronization: Two independent execution paths eventually get synchronized in the generated messaging protocol. Adjustor has to have both a copy of the incident form, from Insurer, as well as a request for cost approval, from Repairer, before inspecting the vehicle. In case the cost approval request is received first, the generated "While" loop with a NOOP ensures that Adjustor never inspects a vehicle until a copy of the incident form is received.

```
Sequence
   Send AccidentLocationResponse From CallCenter To Repairer
   Parallel
      Sequence
         Send IncidentFormResponse From CallCenter To Insurer
         Parallel
            Sequence
               Send IncidentFormCopyResponse From Insurer To Adjustor
               IncidentFormCopyReceived = true
            Sequence
               Send ClaimFormRequest From Insurer To Claimant
               Send ClaimFormResponse From Claimant To Insurer
      Send AssignRepairerResponse From CallCenter To Claimant
   Send CostApprovalRequest From Repairer To Adjustor
   While (NOT IncidentFormCopyReceived)
       NOOP
   Send ReviewedCostEstimateResponse From Adjustor To Insurer
   Send CostApprovalResponse From Adjustor To Repairer
   Send InvoiceResponse From Repairer To Adjustor
   Send ReviewedInvoiceResponse From Adjustor To Insurer
   Send PaymentResponse From Insurer To Repairer
```

Figure 8.9 ACDL description for the original AGFIL case

8.2.2 Requirements for Vehicle Repair – Real World Documents

We analyzed public documents published by government agencies that regulate vehicle insurance and repair business in four of the most populous regions in North America: State of California Department of Insurance (CA)[1]; State of New York Department of Insurance (NY)[2]; State of Illinois Department of Insurance (IL)[3], and the Financial Services Commission of Ontario (FSCO)[4], Canada. From these documents we extracted several requirements that the specification of AGFIL case does not satisfy. These requirements are listed[5] in Table 8.1:

Table 8.1 Requirements extracted from public documents not satisfied by AGFIL case study

ID	Description
CA1	If further damage is found during the repair process, repair shop should contact insurer to get additional cost of repairs approved. The insurer may send out an adjuster to re-inspect additional damages.
CA2	If you do not hear from anyone, call your insurance company for assistance. If they are not responsive, or you believe there is an unreasonable delay in settling your claim, contact the Department of Insurance.
CA3	In the event that you do not agree with your insurance company on the amount of loss either of you can demand an appraisal. Each party selects a competent appraiser. The appraisers then select an umpire.
CA4	The insurance company must stand behind the repairs of the recommended shop if the vehicle is not repaired properly.
FSCO1	When you file a claim for damage or loss, the payment made by the insurance company may be subject to a deductible, i.e. the amount of the claim you will be responsible for paying yourself.
FSCO2	You may also be required to complete a claim form, also known as a Proof of Loss form.

[1] http://www.insurance.ca.gov/0100-consumers/0060-information-guides/0010-automobile/You-had-an-accident.cfm
[2] http://www.ins.state.ny.us/auto/2010/auto10.pdf
[3] http://www.insurance.illinois.gov/autoinsurance/auto_own_claim.asp
[4] http://www.fsco.gov.on.ca/english/insurance/auto/after auto accident ENG.pdf
[5] Requirements quoted from original documents, where "you" refers to Claimant and "insurance company" refers to Insurer.

FSCO3	In some cases the adjuster will want to meet with you in person.
FSCO4	As long as your insurance company approves the estimate, you may have your vehicle repaired at the repair shop of your choice.
IL1	You must immediately report all losses directly to your insurance producer or company.
IL2	Your insurance company may ask for several estimates.
IL3	Your insurance company is required to communicate with you within 21 working days after they are notified of the loss.
NY1	Your insurance company is required to obtain from you and your repair shop a "Certification of Automobile Repairs" form, in order to determine the extent to which your damaged car has been repaired.
NY2	If you fail to submit this "Certification of Automobile Repairs" form, your loss settlement on a subsequent loss may be reduced.

The public documents we analyzed contained a slew of other requirements not addressed by the AGFIL use case, yet that can be deemed outside its scope. These additional requirements can be modularized into interactions independent from vehicle repair. Examples include determining who is at fault in the accident, the Insurer's interaction with other Insurers to recover repair cost, the Claimant getting reimbursed for medical expenditure on injuries resulting from the accident, filing a police report, and providing a rental car for the Claimant for the duration of repairs.

8.2.3 Adapting the AGFIL Literature Case to Real-World Requirements

We analyze each of the requirements listed in Table 8.1 and we work out how to apply our adaptation framework to the AGFIL models in order to incorporate each of these requirements. In summary, each of the thirteen requirements fell into one of the following categories:

- 7 requirements were straightforwardly handled by our adaptation framework. That is, nothing special was involved in applying our framework to this set of requirements

than was needed for the medical example. This set includes: CA1, CA4, FSCO1, FSCO2, FSCO3, IL1, and NY1.

- 3 requirements were cumbersome, but not impossible, to handle using our approach because our approach does not provide a representation to support modularization of an interaction into sub-interactions. This set includes CA3, FSCO4, and IL2.

- 2 requirements could not be captured using the representations at hand. IL3 involve a timing constraint that cannot possibly be captured using LTL used by FT. NY2 involved a constraint that spans multiple instances of the choreographed interaction, which neither Tropos nor WS-CDL are well-equipped to support (Austin et al., 2004).

- 1 requirement did not require any explicit change to the interaction specification. CA2 specifies a mechanism by which the global observer (i.e. Department of Insurance), which is not represented as an explicit interacting role, handles disputes.

We outline the application of our adaptation framework to each of these requirements, providing more detail when tackling requirements that involved more decision-making than the rest, and thus were more challenging to the systematic application of the framework.

8.2.3.1 CA1: Discovering Further Damage during Repair

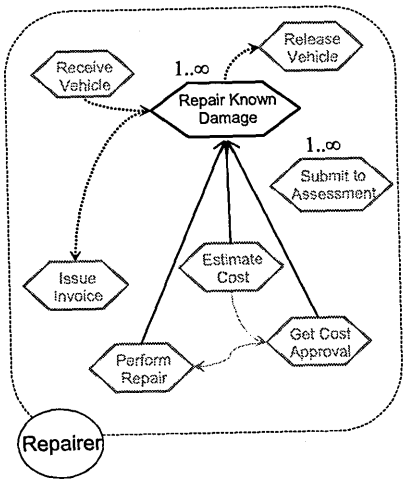
This requirement states that a Repairer must get approval for cost of repairing additional damage discovered during repair. This essentially means that repair may have to be performed over many iterations; during each iteration previously unnoticed damage may be discovered which is repaired in the next iteration, and so on. Whenever new damage is discovered the Repairer is required to get approval from the Adjustor for the new cost.

1	(AL 7.3.2.2 Adapt Refinement) The Repairer identifies activities that constitute a "repair iteration", which are: "Perform Repair", "Estimate Cost", and "Get Cost Approval". For modularity, Repairer designates a new activity, "Repair Known Damage", as a common parent for the three activities to aggregate them together. The fulfillment condition of the

	new activity is that no more damage is found. (AL 7.3.2.4 Adapt Activity Lifecycle) Since the "Repair Known Damage" activity may repeat, it is annotated as being repeating and its repetition propagates to its three children.
2	(AL 7.3.2.3 Adapt Precedence) Realizing that a vehicle should not be released until ALL repairs are done, Repairer adds a precedence link from "Repair Known Damage" to "Release Vehicle" to replace the link that existed from "Perform Repair" to "Release Vehicle". Similarly, a precede link is added from "Repair Known Damage" to "Issue Invoice" in lieu of the one that existed from "Perform Repair".
3	Changes to activity repetition in Repairer local view propagate to the global view through dependencies. (PG 7.4.1.2 Propagate Adaptation of Activity Repetition) The many repetition of "Get Cost Approval" propagates to its associated "Cost Approval" dependency. (AG 7.3.1.3 Adapt Dependency Repetition) The regulatory agency agrees to the suggested change and "Cost Approval" dependency is annotated as being repeating.
4	Changes to the global view propagate to the local view of the depender in "Cost Approval". (PL 7.4.2.3 Adapt Activity Lifecycle) Repetition of "Cost Approval" propagates to the local view of Adjustor who agrees conducting multiple assessments. (AL 7.3.2.4 Adapt Activity Repetition) Adjustor marks "Conduct Assessment" as being repeating. Its repetition propagates to its children, "Inspect Vehicle" and "Approve Estimate".
5	Changes to Adjustor local view propagate back to the global view.

	(PG 7.4.1.2 Propagate Adaptation of Activity Repetition) Repetition of "Inspect Vehicle" propagates to associated dependency, "Present Vehicle". (AG 7.3.1.3 Adapt Dependency Lifecycle) The regulatory agency agrees to the suggested change and the "Present Vehicle" dependency is annotated as being repeating.
6	Changes to the global view propagate to the local view of the dependee in the "Present Vehicle" dependency. (PL 7.4.2.3 Propagate Adaptation of Dependency Repetition) Repetition of "Present Vehicle" propagates to its dependee activity as Repairer agrees to present a vehicle multiple times for assessment. (AL 7.3.2.4 Adapt Activity Repetition) The dependee in "Present Vehicle", which is "Submit to Assessment", is marked as being repeating.
7	Similarly, the repetition annotation of "Approve Estimate" in step 4 is propagated to the dependency "Reviewed Cost Estimate" in the global view and then to "Accept Assessment" activity in Insurer's local view.
8	All participants agree on adapted interaction specification.

The summary of changes made to Repairer's local model, where most adaptations were made, and the ACDL generated from the adapted model is shown in Figure 8.10.




```
Sequence
   Send AccidentLocationResponse From CallCenter To Repairer
   Parallel
      Sequence
         Send IncidentFormResponse From CallCenter To Insurer
         Parallel
            Sequence
               Send IncidentFormCopyResponse From Insurer To Adjustor
               IncidentFormCopyReceived = true
            Sequence
               Send ClaimFormRequest From Insurer To Claimant
               Send ClaimFormResponse From Claimant To Insurer
      Send AssignRepairerResponse From CallCenter To Claimant
   While (NOT NoMoreDamageFound)
      Sequence
         Send CostApprovalRequest From Repairer To Adjustor
         While (NOT IncidentFormCopyReceived)
            NOOP
         Send ReviewedCostEstimateResponse From Adjustor To Insurer
         Send CostApprovalResponse From Adjustor To Repairer
   Send InvoiceResponse From Repairer To Adjustor
   Send ReviewedInvoiceResponse From Adjustor To Insurer
   Send PaymentResponse From Insurer To Repairer
```

Figure 8.10 Adaptation to perform many repair iterations

As expected, the repetition resulted in the "While" construct in the messaging specification. Notice also how the Adjustor performs redundant checks for the receipt of the incident form every time they are sent a cost estimate. This can be avoided by separating the check into its own activity but the Adjustor may choose to do the check every time as part of approving the cost.

8.2.3.2 CA2: Reporting Complaints to Department of Insurance

This requirement specifies that the Claimant has the right to contact the department of insurance to complain about unreasonable delays in claim processing. As specified, this requirement is not incorporated in a choreographed protocol between the participants per se, but rather is representative of how a global observer gets involved indirectly in regulating an interaction. The global observer does not monitor every message exchanged, but rather participants report non-compliance to the global observer who then investigates and prompts non-compliant participants

8.2.3.3 CA3: Involving an Appraiser and an Umpire for Arbitration

If Claimant and Insurer do not agree on estimates, they each select an appraiser who will conduct additional investigations. In case appraisers also do not agree on estimates, they choose an umpire who will have the final say. Disagreement on estimate constitutes a failure of participants to carry out the vehicle repair interaction to completion. This failure triggers another

interaction with different roles and a separate messaging protocol. An "arbitration" interaction takes place between the roles Appraiser, Umpire, Insurer, and Claimant. Although this interaction can be designed separately from the main vehicle repair interaction, integrating them both together is a non-trivial task. This calls for systematic techniques for composing interactions specified using CLGs, which we do not address.

8.2.3.4 CA4: Insurer Stands Behind Repairs if Vehicle is not Repaired Properly

This requirement stipulates that the Insurer is responsible for repairing the vehicle to Claimant's satisfaction. Since this requirement is not specified in detail we will assume that it will be up to the Claimant to declare their dissatisfaction with repairs when they visit the Repairer to pick up the vehicle. That is, Claimant will refuse to pick up the vehicle if they are not satisfied with repairs. That is, the "Repaired Vehicle" dependency is made dismissible which means Repairer may have to repeat the activities "Release Vehicle", "Perform Repairs", "Estimate Cost", "Get Cost Approval", and "Issue Invoice". These will propagate to Adjustor's local view which in turn may repeat "Conduct Assessment", and adaptation continues in a manner similar to that of CA1.

8.2.3.5 FSCO1: Charging Claimant for Deductible

Under this requirement, a Claimant is obliged to pay the difference between what a Repairer charges and what an Insurer pays for repairs. It is left unspecified who charges the Claimant for the deductible so we will assume it is the Repairer, which is the likely alternative.

1	Repairer needs to adapt their local view to charge Claimant for deductible. (AL 7.3.2.1 Add Activity) Repairer adds a "Collect Deductible" activity to their local view. Its creation condition is that they have received payment from Insurer, so that they know how much a Claimant owes. Its fulfillment condition is that deductible has been collected, which we'll refer to as Ω. (AL 7.3.2.3 Adapt Precedence) Repairer considers ways of enforcing the creation condition of "Collect Deductible". They identify that "Receive Payment" brings about the

	creation condition of "Collect Deductible" and hence they add a precede link from "Receive Payment" to the newly added activity.
2	Changes to the local view of the Repairer propagate to the global view. (PG 7.4.1.1 Propagate an Added Activity) The newly added "Collect Deductible" relies on Claimant to provide payment and hence a new dependency needs to be added. (AG 7.3.1.1 Add Dependency) The regulatory agency agrees to a new "Deductible" dependency. Since Claimant fulfills this dependency he is made the dependee. Since the deductible amount is provided to the Claimant by the Repairer, the dependency is annotated as bidirectional.
3	Changes to the global view propagate to the local view of Claimant. (PL 7.4.2.1 Propagate Added Dependency) Claimant accepts the responsibility of paying the deductible amount as being the dependee in the "Deductible" dependency. (AL 7.3.2.1 Add Activity) Claimant adds a “Pay Deductible” activity to be the dependee in the “Deductible” dependency. Since “Deductible” is a bidirectional dependency, the creation condition of the dependee activity is that "Deductible" has been created and its fulfillment is that the deductible amount has been paid, i.e. Ω.
4	As an extension to the analysis performed in step 1, Repairer inspects the interaction specification and decides on further changes. (AL 7.3.2.3 Adapt Activity Precedence) Repairer realizes that it is in their interest not to release a vehicle to a Claimant until they have collected the deductible. Hence, they add precedence from “Collect Deductible” to “Release Vehicle”.
5	(PG 7.4.1.3 Propagate Adaptation of Activity Precedence) The added precedence changes the creation condition of “Release Vehicle” which needs to propagate to the global view

	via the associated "Repaired Vehicle" dependency. (AG 7.3.1.3 Adapt Dependency Lifecycle Condition) Assuming that the Department of Insurance finds it legal to withhold the vehicle, Ω is added as a necessary creation condition for dependency "Repaired Vehicle". That is, paying the deductible becomes a pre-condition on releasing the repaired vehicle.
6	Changes to the "Repaired Vehicle" dependency propagate to the local view of the depender. (PL 7.4.2.3 Propagate Adaptation of Dependency Lifecycle) The change in the creation condition of "Repaired Vehicle" propagates to "Pick up Vehicle" which is the depender activity in Claimant local view. (AL 7.3.2.4 Adapt Activity Lifecycle) Claimant looks for ways to enforce Ω as a necessary pre-condition on the fulfillment of "Pick up Vehicle". Since "Pay Deductible" is already responsible for bringing about Ω, Claimant decides to add precedence from this activity "Pick up Vehicle", thereby ensuring the fulfillment of the former precedes fulfillment of the latter.
7	Participants agree to the adapted interaction specification.

The adapted elements of the model are shown in Figure 8.11. Generating the ACDL from the adapted model results in two additional messages at the end of the interaction that directly follow receipt of payment by the Repairer.

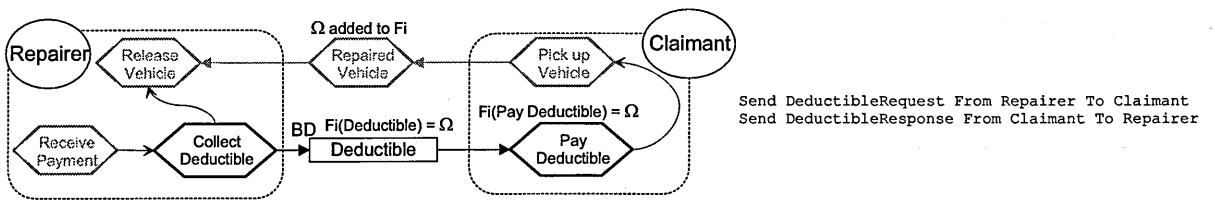


Figure 8.11 Adaptation of vehicle insurance example to incorporate deductible

8.2.3.6 FSCO2: Filling an Claim Form Upon Request

This requirement stipulates that an Insurer "may" require a Claimant to fill a claim form. That is, "Request Claim Form" activity is no more a necessary part of the interaction as Insurer may chose not to execute it. Although the Tropos models do not support annotating an activity as "optional" per se, "OR" refinement may be used to capture this requirement, albeit with undue bloat to the model: Insurer adds a "Process Claim" activity, which is refined into two mutually exclusive activities: "Request Claim Form" (which already exists in the model) and an activity whose semantics are "Process Claim without Requesting Claim Form".

8.2.3.7 FSCO3: Meeting the Adjustor in Person

This requirement gives Adjustor the right to request to meet in person with Claimant, and also get a sworn declaration from them about the accident. Conceivably, Adjustor will call Claimant and suggest a date and place for their meeting. Adjustor and Claimant will then show up at the agreed date and place to discuss the claim. This is realized by adding two dependencies: "Meeting Info" from Claimant to Adjustor and "Appear for Meeting" from Adjustor to Claimant, where the former dependency precedes the latter. To propagate this change to local models of Claimant and Adjustor activities are added at both ends of newly added dependencies. Adaptation of both local models proceeds as has been demonstrated before.

8.2.3.8 FSCO4: Claimant Chooses a Repair Shop

This variant of the vehicle repair interaction appeared in all public documents we reviewed. In this variant Claimant is responsible for hauling their vehicle to a Repairer, obtaining an estimate, and sending it to Adjustor. Even though applying our adaptation framework to this requirement will not be fundamentally different than examples we have discussed before, it is not obvious that starting from the model of Figure 8.8 is beneficial. Arguably, this variant of the vehicle repair interaction is sufficiently different that it may be easier to start from a clean slate. The step-by-step nature of our adaptation process makes it more suited for performing incremental changes but not as effective for larger-grained ones.

8.2.3.9 IL1: Claimant Reports Damages Immediately to Insurer

This requirement stipulates that Claimant should submit a claim form as soon as they can, instead of waiting till Insurer requests submission of claim. This amounts to changing the "Claim Form" dependency from being bidirectional to unidirectional, as Claimant does not rely on Insurer for requesting its fulfillment. Insurer will then change the name of "Request Claim Form" activity to "Receive Claim Form". No other changes are necessary.

8.2.3.10 IL2: Claimant Required to Provide Multiple Estimates

This requirement gives an Insurer the right to request a Claimant to provide multiple repair estimates from a number of licensed repairers. This requirement is similar to FSCO4 in that a Claimant chooses a repairer. The repetition aspect of this requirement can be handled in the same manner as we did for CA1. However, there is a non-trivial challenge with representing multiple instances of the Repairer role. Since a Claimant obtains estimates from multiple repairers, the Repairer role is instantiated multiple times in the same instance of the interaction. WS-CDL does not directly support representing multiple instances of a role nor does ACDL or Tropos diagrams. One solution to this problem is to modularize a repeating part of an interaction as a sub-interaction which is instantiated multiple times. That is, the part of vehicle repair interaction where a Claimant interacts with Repairer is captured in a separate modular "Obtain Estimate" that accepts a repairer instance as a parameter. WS-CDL supports sub-choreographies and ACDL should be extended with an equivalent construct.

8.2.3.11 IL3: Insurer Required to Respond to Claimant within 21 Days

This requirement specifies a maximum time within which an Insurer must respond to Claimant. We found that specifying a maximum time within which a participant is required to respond a commonly occurring requirement. Linear Temporal Logic does not support specifying time limits on events, so formal specification of this requirement is not possible using Formal Tropos. However, WS-CDL does support timeouts and to make use of this feature an extension to Tropos is necessary, potentially via annotating dependencies with a time limit.

8.2.3.12 NY1: Obtaining a Certificate of Repairs from Claimant

This requirement stipulates that a Claimant must obtain a "Certificate of Repairs" from Repairer after repairs are completed and submit it to Insurer. This requirement is captured by adding two dependencies: a "Certificate of Repairs" dependency from Claimant to Repairer to represent the certificate that a Claimant obtains from a Repairer and "Certificate of Repairs Copy" dependency from Insurer to Claimant to represent the copy that a Claimant provides to an Insurer. Since this document may only be produced after repairs are done, a precedence link is added from "Repaired Vehicle" dependency to "Certificate of Repairs" dependency.

8.2.3.13 NY2: Failing to Fill Certificate of Repairs Results in Reducing Next Settlement

If a Claimant fails to submit a Certificate of Repairs an Insurer is left uncertain about the state of the vehicle. If the same Claimant is involved in another future vehicle this requirement gives the Insurer the right to reduce the amount they pay for repairs. An Insurer needs to keep track of repair history in order to check, when a new claim is received, whether a Claimant had submitted a Certificate of Repairs for previous repairs. That is, execution of an interaction instance affects the execution of another, which is not supported in choreography languages as interaction instances are assumed to be independent.

8.2.4 Discussion

We reflect on the results of applying our adaptation framework to the vehicle repair case study and highlight the main findings grouped into the following categories.

8.2.4.1 Capturing and Adapting Requirements

We demonstrated how our adaptation framework supported adaptation driven by local needs as well as those imposed by the regulatory agency. By applying guidance provided by our framework, we propagated changes systematically to all affected parts of a relatively large model. Using our adaptation framework, we were able to handle a majority of requirements found in public documents. The case study exercised a range of adaptation operations of activities, dependencies, and their annotations. The classes of requirements that our framework handles can be expanded via some minor extensions to Tropos annotations. Annotating

dependencies with a timeout, marking an activity as optional, and annotating physical dependencies as dismissible were identified as potential extensions. However, our framework remains better-suited for handling incremental changes and less so for larger-grained changes.

8.2.4.2 Building Valid Models

Building valid models can become harder as the models get larger. In earlier attempts to specify the vehicle repair interaction, we made a mistake by adding a precedence link from one activity to another, where the two activities belonged to mutually exclusive execution branches. Before it starts executing, the activity on the second branch waits for the activity on the first branch to finish. However, the first activity never gets to execute if the second branch is taken, thereby leading to a deadlock. Although these specification problems can be caught by performing model checking after each adaptation operation, an incremental checking technique would save architects time. Furthermore, validity alone is not sufficient for deriving a detailed messaging protocol from a requirements model. To get a detailed messaging specification, dependencies have to be specified at a fine-grained level. For instance, if the relation between Insurer and Adjustor were to be represented using a single "Supervise Repairs" dependency, the generated messaging protocol would have been very terse.

8.2.4.3 Refining Requirements

Requirements in public documents, such as CA4, are not always at a level of refinement suitable for incorporation directly into a CLG. Refinement of high-level requirements into detailed specification, i.e. specific activities and constraints, is outside the scope of our contributions. An analyst should use Tropos or other means for refinement before applying our adaptation framework to incorporate resulting activities and constraints into a CLG. Similarly, refinement of coarse-grained dependencies into finer-grained dependencies appearing in a CLG is a non-trivial task. A high-level RD diagram for the vehicle repair interaction would include a goal dependency from Claimant to Insurer for covering cost of repair. Refinement of this dependency into dependencies and constraints appearing in a refined RD diagram require methodological support such as described by Bresciani (2004).

8.2.4.4 Specifying Interaction Context

In a textual description of an interaction, such as in public documents, it may not be immediately obvious what constitutes local vs. global viewpoints. The original AGFIL specification focused on the business process of the insurance company and its partners, and omitted the Claimant role, which is an essential role for the global view. On the other hand, the public documents did not address in detail the relation that an Insurer has with an Adjustor, such as payment for the Adjustor's service, and hence it was deemed outside the scope of the repair interaction. These findings emphasized the benefit of the global view in specifying interaction context in terms of participating roles and dependencies that are part of the interaction.

The relevance of the global model also derives from the importance of the global observer. Invariably, the public documents specified a dispute resolution service provided by the Department of Insurance. A Claimant who disagrees with an Insurer on interpretation of an insurance policy can file complaints through this service. Although an interaction protocol is necessary for regulating an interaction, it is only part of regulatory material. A range of legal details about how to determine who is at fault in the accident, how to estimate a cash value of the vehicle, etc. cannot be captured in a messaging protocol or even a CLG, but yet should be considered as part of the interaction context.

Specifying interaction context becomes complicated if the set of interacting roles is not fixed. Some of the public documents specified that a Claimant may report an accident to an insurance "broker" which takes care of carrying out the claim process on behalf of the Claimant. This variation complicates the construction of a CLG as a Broker role is only optionally involved. Specification of interacting roles also turned out to be problematic when multiple instances of a role may get involved in the same interaction instance, e.g. in IL2 where multiple instances of Repairer were involved.

8.2.4.5 Validation and Verification via Generation of Messaging Specification

Whereas Tropos models are more suited for requirements-driven adaptation of an interaction, generated ACDL protocols proved to be a useful complementary specification for discovering

errors and omissions during construction of Tropos models. Our ACDL generation techniques provided useful feedback both on the validity of requirements model as well as their consistency with the original AGFIL specification. In earlier attempts to construct the vehicle repair CLG, we failed to capture a necessary precedence link. Our CHREQ tool alerted us that some activities in the model were never executed, which prompted us to look for specification errors. By analyzing constraints governing execution of the activities in question, we were able to attribute the problem to omission of links. The generated messaging specification also helped identify parts of the CLG that were inconsistent with original AGFIL requirements. In one iteration, we specified a precedence link from "Estimate Cost", rather than from "Get Cost Approval", to "Perform Repairs". The generated messaging appeared under-constrained as it allowed an invoice to be sent before cost was approved. This error lead us to look for and identify constraints missing from our CLG governing execution of the activities in question.

8.3 Criteria-Based Evaluation of Adaptation Framework

The service adaptation literature provides a variety of qualitative criteria for evaluating the range of applicability of adaptation approaches. We evaluate our approach using a set of criteria covering scenarios for choreography specification, classes of changes to service interactions, and catalogues of adaptation operations.

8.3.1 Choreographed Specification Scenarios

Section 3.2 discussed approaches that derive local views from a global specification and those that construct a global view from a set of local views. Our approach supports bidirectional change propagation between the global and local views and hence supports both modalities. An additional scenario is described by Decker & von Riegen (2007) where three scenarios for choreography specification were proposed. First, "Choreography identification" proceeds from existing local models to establishing a choreography specification between participants, which we do support via change propagation from local to global view using dependencies. Second, "Choreography Context Expansion" broadens the business context of a choreography specification by incorporating or removing assumptions. We have demonstrated via the vehicle

insurance case study that we support this scenario. However, we do not provide modularization or explicit pre-defined variability points. Third, "Collaboration Unification" obtains a choreography specification by merging existing choreographies. We only provide minimal support for this scenario via incremental adaptation, but not via composition of whole interaction specifications. We also have not dealt with changes in interacting roles.

8.3.2 Classes of Changes to Service Interactions

The nature of service changes may be classified into "shallow", where changes are strictly localized to a service and its clients, or "deep", where the effect of changes cascades beyond to service clients (Papazoglou, 2008). The guidance we provide for adapting global and local views addresses the former whereas the guidance we provide on change propagation addresses the latter. Additionally, changes are further classified as pertaining to structure, business protocol, policies, and operational behavior. As previously discussed, we cover the first three classes. Operational behavior requires consideration of message structure and semantics, which we do not address. Similarly, three classes of changes to business requirements are identified by Desai et al. (2009), namely: transactional (exchanges between participants), structural (organization of participant business processes and delegation between them), and contextual (marketplace rules and government regulations). We have demonstrated how we support these three requirements as changes to global model, local model, and delegation between models via dependencies. Furthermore, our particular usage of CLG diagrams addresses the criticism that Tropos has only been used to model and adapt centrally-orchestrated systems.

8.3.3 Catalogues of Adaptation Operations

Thirteen patterns of business process change were identified by Weber et al. (2007). In addition to supporting the basic patterns of inserting or deleting elements, our approach also supports more advanced ones such as making part of a process conditional, e.g. via OR-refinement or adapting lifecycle conditions. However, we do not provide guidance on supporting "refactoring" operations of a process such as extracting a sub-process.

Our approach is consistent with adapting Tropos model in terms of additions and removal of elements (Krishna et al., 2004), albeit we do not address adding or removing roles. This view is also consistent with that of Mens & D'Hondt (2000) where a model is adapted via addition, removal, connection, and disconnection of model elements. Additionally, compositions of those elementary changes are considered, for which we do not provide specific guidance except via change propagation. However, these sets of operations do not include changes to lifecycle conditions, whereas our set of operations does.

8.4 Chapter Summary

In this chapter we evaluated our adaptation framework using two case studies. The first utilized the familiarity of the constructed medical example to demonstrate the applicability of our approach and provided resolution to the challenges presented in section 2.4. The second case study drew on public government documents and a well-studied vehicle insurance scenario to demonstrate the utility of our approach in a real-world context. Other than justifiable assumptions about stakeholder intentions, we applied our adaptation framework in a systematic manner repeatable by an architect who has basic experience with Tropos modeling. We were able to apply our guidance framework to address a majority of the requirements that we encountered. We also demonstrated how our ACDL generation technique is useful in verification and validation of requirements models. We concluded our evaluation by judging the range of utility of our approach versus a set of criteria drawn from the service adaptation literature. Throughout the evaluation we have identified some limitations of our approach to be discussed in the next chapter.

Chapter 9. Summary and Further Work

This thesis tackled the problem of adapting choreographed service-oriented interaction protocols. We identified three challenges in the way of solving this problem. Firstly, state-of-the-art choreography description standards are detached from the business goals of interacting participants, which makes it hard to ensure adaptations of an interaction protocol satisfy these goals. Secondly, adapting a choreographed interaction requires agreement between multiple stakeholders, which calls for means to resolve their potentially conflicting needs. Thirdly, the space of alternatives for adapting an interaction is typically large, and so ad hoc navigation of this space is inefficient and error-prone. Our contributions towards solving this problem, summarized in section 9.1, directly tackle these three challenges. We discuss the limitations of our approach and future work in section 9.2, and finally conclude in 9.3.

9.1 Summary of Contributions

Incorporating emergent requirements into the specification of a cross-organizational interaction is a multi-faceted problem. Stakeholder goals driving them to engage in an interaction as well as global requirements may evolve separately. Adaptation of an interaction protocol needs to satisfy both sets of interaction requirements while maintaining consistency between the global view and the local views of interaction specifications. We adopted a requirements-driven approach that provides a methodological path from problem to solution. We presented three contributions that directly address the three identified challenges.

Firstly, we represented a choreographed interaction using models of organizational requirements motivating the interaction. Organizational requirements models embody business goals as well as global constraints imposed by the context, thereby enabling business-level reasoning about adaptation. We exploited the semantics of requirements models to automate the derivation of messaging protocols from them, thereby bridging the gap in level of abstraction between requirements specification and messaging specifications. This contribution brings

together two fields: Requirements Engineering and SOC. On the one hand, by advocating requirements-driven adaptation of choreographed interactions we bring the benefits of Requirements Engineering (RE) to SOC. On the other hand, by grounding Tropos in choreography we extend the reach of the methodology and bring it closer to the realm of standards offered by SOC.

Secondly, we established relations between the viewpoints that represent the interests of stakeholders in an interaction. We bridge the local business needs of each participant in an interaction to the global constraints imposed on the interaction, thereby enabling each stakeholder to guard their interests while fulfilling their obligations. By propagating changes across viewpoints we maintain consistency between them and drive the forging of an agreement between stakeholders on an adapted interaction specification. This contribution complements the Tropos methodology with a formalization for separating/integrating concerns in a multi-participant choreographed system.

Thirdly, we proposed a framework for guiding adaptation of interaction specifications. The framework helps stakeholders ensure the validity of an interaction specification being adapted. We built a catalogue of adaptation operations that guides architects through the process of adapting an interaction specification. For each adaptation operation, we provide guidance on how to apply it as well as how to traverse the space of available adaptation alternatives. The catalogue guides systematic change propagation between viewpoints to maintain consistency between them. Our proposed adaptation process backed by the guidance of the catalogue provides a blueprint for implementing tool-assisted adaptation of Tropos models.

9.2 Further Work

In chapter 3 we set the stage of our work thereby deeming some work to be outside the scope of this thesis. During evaluation of our approach, further limitations were identified which we discuss here along with potential means for addressing them. We outline further work to extend our approach and render it useful in a wider context.

9.2.1 Modularizing Representation of Choreographed Interaction

As seen in Figure 8.8 graphical Tropos models for specifying an interaction get complex pretty quickly. It is desirable to break a Tropos model down into smaller fragments that are easier to comprehend and manage. Partitioning Tropos models would also help generate a modular interaction protocol, where reusable blocks of the protocol are modularized into sub-choreographies, a construct that WS-CDL does support. This is particularly useful when an interaction contains many alternative paths that make the "main flow" harder to comprehend. Partitioning a model would also help support multiple instances of a role per interaction, a common problem in choreographed specification (Decker et al., 2008). For instance, in the vehicle repair scenario, to specify that a Claimant may receive competing estimates from multiple Repairers, we may specify a separate interaction between Claimant and Repairer as a sub-interaction out of which many instance may be created per instance of the main interaction.

To get an integrated view of an interaction from Tropos model fragments we thus need means for composing models. Although our adaptation framework guides incorporation of elements (e.g. dependency and activity) into an existing model, it does not solve the general problem of composing Tropos model fragments. Composing multi-viewpoint Tropos models requires dealing with the potential for inconsistencies, a problem that has been tackled in the requirements engineering literature (Ghezzi & Nuseibeh, 1998, 1999). In this respect, existing Tropos modeling tools (Bertolini et al., 2006) are pretty limited and better tooling support is called for. Commitments-based approaches, as was discussed in sections 3.1.1.4, provide a business-level representation of participant inter-connections. One notable advantage of these approaches is the systematic construction of an interaction protocol by composing reusable protocols (Desai et al., 2005). Combining this flexibility with the business-level representation offered by Tropos is a potential approach to modularize representation of choreographed interactions (Telang & Singh, 2009).

9.2.2 Representing Timing Constraints

The vast majority of constraints found in the literature can be captured in terms of occurrence, or absence, of events and precedence between them (Dwyer et al., 1999); this also applies to

service-oriented systems (Li et al., 2005). However, we have encountered some fundamental classes of constraints that cannot be easily represented using these primitives or any other Linear-time Temporal Logic constructs. Constraints requiring a participant to respond within a specified time window, i.e. time-bounded responses, stand out as one. Formalisms other than LTL, such as Duration Logic, have been used to represent this class of timing constraints (Kazhamiakin et al., 2006). Also, some extensions of temporal logic allow the expression of constraints such as "condition Φ must eventually hold within the next t time units" (Koymans, 1992). One such extension, Fluent LTL (Giannakopoulou & Magee, 2003), was extended to specify and verify behavior of event-based systems, e.g. message-oriented systems (Letier et al., 2005). Conceivably, Formal Tropos can be upgraded to use a variant of LTL that can capture time-bounded responses. On the messaging specification-side, WS-CDL does allow specifying timeouts for interaction events, so it can already accommodate these constraints.

9.2.3 Handling Service Failures

In a distributed messaging system failures occur due to faulty communication channels or due to business-level failure. A business-level failure occurs when a participant fails to comply with their obligations, either by providing an invalid or untimely response (Cristian, 1991). Channel failures are typically handled by superimposing reliable messaging mechanisms such as retrying to send failed messages (OASIS, 2009). Handling business-level failures is more complex. We have introduced dependency dismissibility as a means for handling failure of a participant to comply with an interaction specification. However, we have not addressed the issue of rolling back a partially complete interaction, an often required recovery technique in distributed systems (Treaster, 2005). Rolling back an interaction is a non-trivial issue, especially without explicit annotation of service properties, most notably whether operations are idempotent (Hobbs et al., 2008). Capturing "exceptional" paths of an interaction, i.e. those executed when a business-level failure occurs, within the same CLG as the "main" path may yield an unreadable specification. This again calls for means to modularize interaction specifications in Tropos.

9.2.4 Building a Two-Way Street between Requirements and Messaging

Deriving messaging protocols from requirements models assumes that requirements models are the single authoritative representation of an interaction specification. This assumption does not take into account scenarios where the only existing specification of an interaction is a messaging protocol. To adapt an interaction in this scenario an architect has to reconstruct requirements models from a messaging protocol prior to performing our adaptation process on requirements models. This reconstruction is non-trivial for the same reasons that we proposed performing adaptation to requirements rather than messaging protocols. Requirements models embody business knowledge as well as physical activities missing from a messaging protocol. Any tool that would support this reverse derivation has to consult a human, or a well-populated knowledge base, in order to succeed in reconstruction. We discussed attempts for relating incremental changes in a BPMN process specification back to requirements models (Koliadis et al., 2006a). Such approaches only consider incremental co-evolution of requirements and business process, but do not tackle scenarios that start from only a messaging protocol. Nevertheless, incremental two-way change propagation between requirements and messaging is a starting point for relaxing the assumption that requirements models are the single authoritative specification.

9.2.5 Extending the Adaptation Process

In section 8.3, we demonstrated that our proposed adaptation operations are consistent with what has been proposed in the adaptation literature. Sequences of add/remove operations on a model are sufficient for performing incremental adaptation of a model to incorporate an emergent requirement. However, our proposed operations are fine-grained and do not benefit from commonly occurring patterns in process-oriented systems such as "Notification", "Approval", and others (Thom et al., 2009). By applying such patterns it may be possible to reduce the number of adaptation steps and minimize chances of errors. It is thus desirable to compose our primitive operations into coarser-grained operations with business-level semantics (Weber et al., 2007). For instance, to specify that a Doctor is allowed to cancel a previously

confirmed appointment, instead of applying a sequence of primitive operations, one would apply the "Revert Offer" operation (Singh et al., 2009).

9.3 Conclusion

This thesis presented three contributions to address the problem of adapting choreographed interaction specifications. The contributions provide a framework for representing an interaction using requirements models, guiding adaptation of these models, enabling collaboration between participants on adaptation, and systematically deriving a choreographed messaging protocol. The utility of the contributions was validated using case studies and criteria from the literature, which helped identify some limitations of our approach. We outlined potential directions for extending our approach to tackle these limitations.

Bibliography

Acharya, M., Kulkarni, A., Kuppili, R., Mani, R., More, N., Narayanan, S., et al. (2005). "SOA in the Real World – Experiences," *in Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands: Springer, Dec 12-15, pp. 437-449

Allen, R. & Garlan, D. (1997). "A Formal Basis for Architectural Connection," *ACM TOSEM*, vol. 6 no. 3, pp. 213-249.

Almeida, J. P., Dijkman, R., Pires, L. F., Quartel, D., & van Sinderen, M. (2005). "Abstract Interactions and Interaction Refinement in Model-Driven Design," *in Proceedings of the International Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2005) co-located with the 9th IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, Enschede, The Netherlands: IEEE Computer Society, 20 September, pp. 273-286

Almeida, J. P., van Sinderen, M., Pires, L. F., & Quartel, D. (2003). "A Systematic Approach to Platform-Independent Design Based on the Service Concept," *in Proceedings of the Enterprise Distributed Object Computing Conference (EDOC'03)*, Brisbane, Australia, 16-19 Sep, pp. 112- 123

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web Services: Concepts, Architectures, and Applications*: Springer Verlag.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., J. Klein, Leymann, F., et al. (2003). "Business Process Execution Language for Web Services Version 1.1," W3C.

Arbab, F. (1996). *Coordination Languages and Models (Proceedings of the First International Conference)*, vol. 1061 (1996). Cesena, Italy: Springer.

Austin, D., Barbir, A., Peters, E., & Ross-Talbot, S. (2004). "Web Services Choreography Requirements," W3C.

Baglietto, P., Maresca, M., Parodi, A., & Zingirian, N. (2002). "Deployment of Service Oriented Architecture for a Business Community.," *in Proceedings of the 6th International IEEE Enterprise Distributed Object Computing Conference (EDOC'02)*, Lausanne, Switzerland, 17-20 Sep, pp. 293-304

Baldoni, M., Baroglio, C., Chopra, A. K., Desai, N., Patti, V., & Singh, M. P. (2009). "Choice, Interoperability, and Conformance in Interaction Protocols and Service Choreographies," *in Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, Budapest, Hungary: IFAAMAS, May 10-15, pp. 843-850

Baldoni, M., Baroglio, C., Martelli, A., Patti, V., & Schifanella, C. (2005). "Verifying the Conformance of Web Services to Global Interaction Protocols: a First Step," *in Proceedings of the 2nd International Workshop on Web Services and Formal Methods (WS-FM)*, Versailles, France: Springer, Sep 1-3, pp. 257-271

Barros, A. & Boerger, E. (2005). " A Compositional Framework for Service Interaction Patterns and Interaction Flows," *in Proceedings of the 7th International Conference on Formal Engineering Methods (ICFEM'2005)*, Manchester, UK: Springer Verlag, November, pp. 5-35

Barros, A., Dumas, M., & Oaks, P. (2005a). "A critical overview of the web services choreography description language (WS-CDL)," *BP Trends*, vol. 3 (March), pp. 1-24.

Barros, A., Dumas, M., & Oaks, P. (2006). "Standards for Web Service Choreography and Orchestration: Status and Perspectives," *in Proceedings of the Business Process Management Workshops*, Vienna, Austria: Springer, Sep 4-7, pp. 61-74

Barros, A., Dumas, M., & ter Hofstede, A. H. M. (2005b). "Service Interaction Patterns," *in Proceedings of the 3rd International Conference on Business Process Management (BPM)*, Nancy, France: Springer Verlag, Sep 6-8, pp. 302-318

Benatallah, B., Casati, F., Grigori, D., Motahari-Nezhad, H. R., & Toumani, F. (2005). "Developing Adapters for Web Services Integration," *in Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE)*, Porto,Portugal: Springer Verlag, June 13-17, pp. 415-429

Benatallah, B., Dumas, M., Sheng, Q. Z., & Ngu, A. H. H. (2002). "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services," *in Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, San Jose, California, USA, March 2002, pp. 297 - 308

Benington, H. D. (1987). "Production of large computer programs," *in Proceedings of the the 9th International Conference on Software Engineering (ICSE)*, Monterey, California, United States: IEEE Computer Society Press, March 30 - April 2, pp. 299-310

Berry, A. & Milosevic, Z. (2005). "Extending Choreography With Business Contract Constraints," *International Journal of Cooperative Information Systems (IJCIS)*, vol. 14 no. 2/3, pp. 131-179.

Berry, D., Kazman, R., & Wieringa, R., Eds. (2003). *Proceedings of the Second International Workshop on From Software Requirements to Architectures (STRAW'03) co-located with the International Conference on Software Engineering (ICSE'03)*. Portland, Oregon: IEEE Computer Society.

Bertolini, D., Delpero, L., Mylopoulos, J., Novikau, A., Penserini, L., Perini, A., et al., "A Tropos Model-Driven Development Environment," in *Proceedings of the18th Conference on Advanced Information Systems Engineering (CAiSE '06), Forum Proceedings, Theme: Trusted Information Systems*. Luxembourg: ScienceStage.com, June 5-9, 2006.

Bertrand, P., Darimont, R., Delor, E., Massonet, P., & van Lamsweerde, A. (1997). "GRAIL/KAOS: An Environment for Goal Driven Requirements Engineering," *in Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, Boston, Massachusetts, USA: IEEE-ACM, May 17-23, pp. 612-613

Boehm, B. (1981). *Software Engineering Economics*: Prentice Hall.

Bohner, S. A. & Arnold, R. S., Eds. (1996). *Software Change Impact Analysis*. Los Alamitos, California, USA: IEEE Computer Society Press.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language User Guide*: Addison-Wesley.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., et al. (2004). "Web Services Architecture," W3C, 11 February 2004.

Bray, T., Hollander, D., Layman, A., Tobin, R., & Thompson, H. S. (2009). "Namespaces in XML 1.0," W3C, December 2009.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2004). "Extensible Markup Language (XML) 1.0 (Third Edition)," W3C.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). "Tropos: An Agent-Oriented Software Development Methodology," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 8 no. 3 (May 2004), pp. 203-236.

Browne, S. & Kellet, M. (1999). "Insurance (motor damage claims) scenario," CrossFlow Consortium Document Identifier D1.a.

Bryl, V., Giorgini, P., & Mylopoulos, J. (2006). "Designing Cooperative IS: Exploring and Evaluating Alternatives," *in Proceedings of the On the Move to Meaningful Internet Systems (OTM Confederated International Conferences): CoopIS*, Montpellier, France: Springer, Oct 29 - Nov 3, pp. 533-550

Burdett, D. & Kavantzas, N. (2004). "WS Choreography Model Overview," W3C Choreography Working Group, 24 March 2004.

Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., & Zavattaro, G. (2005). *in Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands: Springer, Dec 12-15, pp. 228-240

Cai, C., Qiu, Z.-Y., Yang, H.-L., & Zhao, X.-P. (2009). "Global-to-Local Approach to Rigorously Developing Distributed System with Exception Handling," *Journal of Computer Science and Technology*, vol. 24 no. 2, pp. 238-249.

Carbone, M., Honda, K., & Yoshida, N., "Structured Communication-Centred Programming for Web Services," in *Programming Languages and Systems*, vol. 4421, *Lecture Notes in Computer Science*, R. De Nicola, Ed.: Springer Berlin / Heidelberg, 2007, pp. 2-17.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). "Towards Requirements-Driven Information Systems Engineering: The Tropos Project," in *Information Systems*, vol. 27: Elsevier, pp. 365–389.

Cerami, E. (2002). *Web Services Essentials*: O'Reilly and Associates.

Chakravarty, P. & Singh, M. P. (2008). "Incorporating Events into Cross-Organizational Business Processes," *IEEE Internet Computing*, vol. 12 (2008/03/01), pp. 46-53.

Chandy, K. M. & Rifkin, A. (1997). "Systematic Composition of Objects in Distributed Internet Applications: Processes and Sessions," *Computer Journal*, vol. 40 no. 8 (October 1997), pp. 465-478.

Charfi, A. & Mezini, M. (2004a). "Aspect-Oriented Web Service Composition with AO4BPEL," *in Proceedings of the European Conference on Web Service (ECOWS'04)*, Erfurt, Germany: Springer, Sep 27-30, pp. 168-182

Charfi, A. & Mezini, M. (2004b). "Hybrid Web Service Composition: Business Processes Meet Business Rules," *in Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, NY, USA, Nov 15-19, pp. 30-38

Cheng, B. & Atlee, J. (2007). "Research Directions in Requirements Engineering," in *Future of Software Engineering*: IEEE Computer Society Washington, DC, USA, pp. 285-303.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). "Web Services Description Language (WSDL) 1.1," W3C.

Chung, L., Nixon, B. A., & Yu, E. (1995). "Using Non-Functional Requirements to Systematically Support Change," *in Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*, York, UK: IEEE Computer Society, pp. 132-139

Clarke, E. M., Grumberg, O., & Peled., D. (1999). *Model Checking*: MIT Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2002). *Introduction to Algorithms*: MIT Press.

Cristian, F. (1991). "Understanding Fault-Tolerant Distributed Systems," *Communications of the ACM*, vol. 34 no. 2, pp. 56-78.

Cryer, P. (2000). *The Research Student's Guide to Success*, 2nd ed. Buckingham: Open University Press.

CSC (2000). "13th Annual Critical Issues of IS Management Survey," Computer Sciences Corporation.

Curbera, F., Ferguson, D., Nally, M., & Stockton, M. L. (2005). "Toward a Programming Model for Service-Oriented Computing," *in Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands: Springer, Dec 12-15, pp. 33-47

Curbera, F., Khalaf, R., Mukhi, N., Tai, S., & Weerawarana, S. (2003). "The Next Step in Web Services," *Communications of the ACM*, vol. 46 no. 10, pp. 29-34.

Dam, K. H. & Winikoff, M. (2004). "Comparing Agent-Oriented Methodologies," in *Agent-Oriented Information Systems*, vol. 3030, P. Giorgini, B. Henderson-Sellers, & M. Winikoff, Eds.: Springer-Verlag, pp. 78-93.

Dam, K. H., Winikoff, M., & Padgham, L. (2006). "An Agent-Oriented Approach to Change Propagation in Software Evolution," *in Proceedings of the Australian Software Engineering Conference*, Melbourne, Victoria, Australia: IEEE Computer Society, April 18-21, pp. 309-318

Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). "Goal-directed Requirements Acquisition," *Science of Computer Programming*, vol. 20 no. 1-2, pp. 3-50.

Darimont, R. & van Lamsweerde, A. (1996). "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," *in Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, San Francisco, California, USA: ACM Press, Oct 16-18, pp. 179-190

De Landtsheer, R., Letier, E., & van Lamsweerde, A. (2004). "Deriving Tabular Event-Based Specifications from Goal-Oriented Requirements Models," *Requirements Engineering*, vol. 9 no. 2 (May), pp. 104-120.

Decker, G. & Barros, A. (2008). "Interaction Modeling Using BPMN," *in Proceedings of the Business Process Management Workshops*, Milano, Italy: Springer, Sep 1-4, pp. 208-219

Decker, G., Kopp, O., & Barros, A. (2008). "An Introduction to Service Choreographies," *Information Technology*, vol. 50 no. 2, pp. 122-127.

Decker, G., Overdick, H., & Zaha, J. M. (2006a). "On the Suitability of WS-CDL for Choreography Modeling," *in Proceedings of the Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen (EMISA 2006)*, Hamburg, Germany: Gesellschaft für Informatik, Oct 17-18, pp. 21-33

Decker, G. & von Riegen, M. (2007). "Scenarios and Techniques for Choreography Design," *in Proceedings of the 10th International Conference on Business Information Systems*, Poznan, Poland: Springer-Verlag, Apr 25-27, pp. 121-132

Decker, G., Zaha, J. M., & Dumas, M. (2006b). "Execution Semantics for Service Choreographies," *in Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM'06) co-located with the 4th International Conference on Business Process Management (BPM 2006)*, Vienna, Austria: Springer, Sep 8-9, pp. 163-177

DeLine, R. (1999). "A Catalog of Techniques for Resolving Packaging Mismatch," *in Proceedings of the Symposium on Software Reusability*, Los Angeles, California, USA: ACM Press New York, NY, USA, pp. 44 - 53

Desai, N., Chopra, A. K., & Singh, M. P. (2006). "Business Process Adaptations via Protocols," *in Proceedings of the IEEE International Conference on Services Computing, 2006 (SCC '06)*, Chicago, IL, USA: IEEE Computer Society, 18-22 Sep, pp. 103-110

Desai, N., Chopra, A. K., & Singh, M. P. (2007). "Representing and Reasoning About Commitments in Business Processes," *in Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*, Vancouver, British Columbia, Canada: AAAI Press, July 22-26, pp. 1328-1333

Desai, N., Chopra, A. K., & Singh, M. P. (2009). "Amoeba: A Methodology for Modeling and Evolution of Cross-Organizational Business Processes," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 19 no. 2 (April 2010).

Desai, N., Mallya, A. U., Chopra, A. K., & Singh, M. P. (2005). "Interaction Protocols as Design Abstractions for Business Processes," *IEEE Transactions on Software Engineering*, vol. 31 no. 12, pp. 1015-1027.

Desai, N. & Singh, M. P. (2007). "A Modular Action Description Language for Protocol Composition," *in Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, Vancouver, British Columbia, Canada: AAAI Press, pp. 962-967

Dijkman, R. & Dumas, M. (2004). "Service-Oriented Design: A Multi-Viewpoint Approach," *International Journal of Cooperative Information Systems (IJCIS)*, vol. 13 no. 4 (2004), pp. 337-368.

Dijkstra, E. W. (1974). Springer-Verlag, Accessed on 28th Nov 2010, http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html.

Dwyer, M. B., Avrunin, G. S., & Corbett, J. C. (1999). "Patterns in Property Specifications for Finite-state Verification.," *in Proceedings of the International Conference on Software Engineering (ICSE'99)*, Los Angeles, CA, USA: ACM, May 16-22, pp. 411-420

Earl F. Ecklund, J., Delcambre, L. M. L., & Freiling, M. J. (1996). "Change Cases: Use Cases that Identify Future Requirements," *in Proceedings of the 11th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'96)*, San Jose, California, USA: ACM, Oct 6-10, pp. 342-358

Edmonds, E. A. (1974). "A Process for the Development of Software for Nontechnical Users as an Adaptive System," *General Systems*, vol. 19, pp. 215–218.

Emmerich, W., Lamanna, D., Piccinelli, G., & Skene, J. (2003). "Method for Service Composition and Analysis," University College London, Technical Report D3, October.

Erl, T. (2004). *Service-oriented architecture: a field guide to integrating XML and Web services*: Prentice Hall PTR.

Feather, M., Fickas, S., Finkelstein, A., & van Lamsweerde, A. (1997). "Requirements and Specification Exemplars," *Automated Software Engineering*, vol. 4 no. 4 (October), pp. 419-438.

Foster, H. (2006). "A Rigorous Approach To Engineering Web Service Compositions," PhD Thesis, Department of Computing, Department of Computing, Imperial College London - University Of London, London.

Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2006). "Model-Based Analysis of Obligations in Web Service Choreography," *in Proceedings of the AICT-ICIW'06*, Guadeloupe, French Caribbean: IEEE Computer Society, Feb 19-25, pp. 149

Fuxman, A. (2001). "Formal Analysis of Early Requirements Specifications.," Master's thesis, Computer Science, University of Toronto, Toronto - Canada.

Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., & Traverso, P. (2004). "Specifying and Analyzing Early Requirements in Tropos," *RE Journal*, vol. 9 no. 2, pp. 132-150.

Fuxman, A., Liu, L., Pistore, M., Roveri, M., & Mylopoulos, J. (2003). "Specifying and Analyzing Early Requirements: Some Experimental Results," *in Proceedings of the 11th IEEE International Conference on Requirements Engineering*, Kyoto, Japan: IEEE Computer Society, Sep 6-10, pp. 132-150

Fuxman, A., Mylopoulos, J., Pistore, M., & Traverso, P. (2001). "Model Checking Early Requirements Specifications in Tropos," *in Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE '01)*, Toronto, Canada: IEEE Computer Society, Aug 27-31, pp. 174-181

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley.

Garlan, D. (1998). "Higher-Order Connectors," *in Proceedings of the Workshop on Compositional Software Architecture*, Monterey, CA, USA, Jan 6-7

Garlan, D., Allen, R., & Ockerbloom, J. (1995). "Architectural mismatch, or Why it's hard to build systems out of existing parts.," *in Proceedings of the International Conference on Software Engineering (ICSE)*, Seattle, Washington, USA: ACM Press, April 23-30, pp. 179-185

Gehlert, A., Bramsiepe, N., & Pohl, K. (2008). "Goal-Driven Alignment of Services and Business Requirements," *in Proceedings of the International Workshop on Service-Oriented Computing Consequences for Engineering Requirements (SOCCER'08) co-located with the 16th IEEE International Requirements Engineering Conference*, Barcelona, Spain: IEEE, Sep 8, pp. 1-7

Georgakopoulos, D. & Papazoglou, M., Eds. (2008). *Service-oriented computing*, Cooperative Information Systems series vol.: MIT Press.

Ghezzi, C. & Nuseibeh, B., Eds. (1998). *Special Issues on Inconsistency Management in IEEE Transactions on Software Engineering*24(11).

Ghezzi, C. & Nuseibeh, B., Eds. (1999). *Special Section on Inconsistency Management in IEEE Transactions on Software Engineering*25(6).

Ghose, A. & Koliadis, G. (2008). "Actor Eco-systems: Modeling and Configuring Virtual Enterprises," *in Proceedings of the IEEE Congress on Services Part II*, Beijing, China, Sep 23-26, pp. 125-132

Giannakopoulou, D. & Magee, J. (2003). "Fluent Model Checking for Event-Based Systems," *in Proceedings of the 9th European Software Engineering Conference co-located with the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, Helsinki, Finland: ACM, Sep 1-5, pp. 257-266

Giorgini, P., Mylopoulos, J., Nicchiarelli, E., & Sebastiani, R. (2003). "Formal Reasoning Techniques for Goal Models," *Journal on Data Semantics*, vol. 2800, pp. 1-20.

Giorgini, P., Mylopoulos, J., Penserini, L., Perini, A., & Susi, A. (2008). "Tropos at the Age of Eight: On-going Research at FBK, UniTN and UT," *in Proceedings of the 3rd International i* Workshop*, Recife, Brazil: CEUR Workshop Proceedings, February 11-12

Giunchiglia, F., Mylopoulos, J., & Perini, A. (2002). "The Tropos Software Development Methodology: Processes, Models and Diagrams," *in Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, Bologna, Italy: ACM, July 15-19, pp. 35-36

Grefen, P., Aberer, K., Hoffner, Y., & Ludwig, H. (2000). "CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises," *Computer Systems Science & Engineering*, vol. 1 no. 5, pp. 277-290.

Gu, X., Nahrstedt, K., Yuan, W., Wichadakul, D., & Xu, D. (2001). "An XML-based Quality of Service Enabling Language for the Web," Department of Computer Science, University of Illinois at Urbana-Champaign UIUCDCS-R-2001-2212.

He, Q., Yan, J., Jin, H., & Yang, Y. (2008). "Adaptation of Web Service Composition Based on Workflow Patterns," *in Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC)*, Sydney, Australia: Springer-Verlag, Dec 1-5, pp. 22-37

Herrmann, K. (2005). "Self-Management: The Solution to Complexity or Just Another Problem?," *IEEE Distributed Systems Online*, vol. 6 (Jan).

Hiel, M. & Weigand, H. (2009). "Interoperability Changes in an Adaptive Service Orchestration," *in Proceedings of the IEEE International Conference on Web Services (ICWS)*, Los Angeles, CA, USA: IEEE Computer Society, 6-10 July, pp. 351-358

Hinge, K., Ghose, A., & Koliadis, G. (2009). "Process SEER: A Tool for Semantic Effect Annotation of Business Process MModels," *in Proceedings of the 3th IEEE International Conference on Enterprise Distributed Object Computing (EDOC)*, Auckland, New Zealand: IEEE Press, Sep 1-4, pp. 49-58

Hoare, C. A. R. (1985). *Communicating Sequential Processes*: Prentice Hall.

Hobbs, C., Becha, H., & Amyot, D. (2008). "Failure Semantics in a SOA Environment," *in Proceedings of the International MCETECH Conference on e-Technologies*, Montreal (Quebec) Canada: IEEE Computer Society, Jan 23-25, pp. 116-121

Huhns, M. N. & Singh, M. P. (2005). "Service-Oriented Computing Key Concepts and Principles," *IEEE Internet Computing*, vol. 9 no. 1, pp. 75-81.

Huhns, M. N., Singh, M. P., Burstein, M., Decker, K., Durfee, E., Finin, T., et al. (2005). "Research Directions for Service-Oriented Multiagent Systems," *IEEE Internet Computing*, vol. 9 no. 6 (Nov-Dec), pp. 65–70.

Jackson, M. (1990). "Some Complexities in Computer-Based Systems and Their Implications for System Development," *in Proceedings of the International Conference on Computer Systems and Software Engineering (CompEuro '90)*, Tel-Aviv, Israel: IEEE Computer Society Press, May 8-10, pp. 344-351

Jackson, M. (1996). "Connecting Viewpoints by Shared Phenomena," *in Proceedings of the 2nd International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, San Francisco, California, USA: ACM Press, Oct 16-18, pp. 180-183

Jackson, M. (2001). *Problem Frames: Analyzing and Structuring Software Development Problems*: Addison-Wesley.

Josuttis, N. (2007). *SOA in Practice: The Art of Distributed System Design*: O'Reilly Media, Inc.

Jureta, I. J., Faulkner, S., & Thiran, P. (2007). "Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems," *in Proceedings of the International Conference on Service-Oriented Computing (ICSOC 2007)*, Vienna, Austria: Springer, Dec 17-20, pp. 270-282

K.H.Bennett, M.Munro, N.Gold, P.J.Layzell, D.Budgen, & O.P.Brereton (2001). "Achieving Ultra Rapid Evolution Using Service-based Software," *in Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE'01) co-located with the 8th European Software Engineering Conference (ESEC)*, Florence, Italy, Sep 10-14, pp. 292-300

Kavantzas, N., Burdett, D., & Ritzinger, G. (2005). "Web Services Choreography Description Language Version 1.0," W3C.

Kaye, D. (2003). *Loosely Coupled: The Missing Pieces of Web Services*: RDS Press.

Kazhamiakin, R., Pandya, P., & Pistore, M. (2006). "Timed Modeling and Analysis in Web Service Compositions," *in Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES'06)*: IEEE Computer Society, Apr 20-22, pp. 840-846

Kazhamiakin, R. & Pistore, M. (2006). "Analysis of Realizability Conditions for Web Service Choreographies," *in Proceedings of the International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, Paris, France, Sep 26-29, pp. 61-76

Kazhamiakin, R., Pistore, M., & Roveri, M. (2004). "A Framework for Integrating Business Processes and Business Requirements," *in Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC'04)*, Monterey, California, USA: IEEE Computer Society, Sep 20 - 24, pp. 9-20

Khalaf, R., Mukhi, N., & Weerawarana, S. (2003). "Service-Oriented Composition in BPEL4WS," *in Proceedings of the 12th World Wide Web Conference, Web Services Track*, Budapest, Hungary: ACM, May 20-24

Khaled, M. & Spanoudakis, G. (2004). "A Framework for Requirements Monitoring of Service Based Systems," *in Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, New York, USA, Nov 15-19i, pp. 84-93

Kiczales, G. (1997). "Aspect-Oriented Programming," *in Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Jyväskylä, Finland: Springer-Verlag, June 9-13, pp. 220-242

Kohlborn, T., Korthaus, A., Chan, T., & Rosemann, M. (2009). "Identification and Analysis of Business and Software Services - A Consolidated Approach," *IEEE Transactions on Service Computing*, vol. 2 no. 1, pp. 50-64.

Koliadis, G., Vranesevic, A., Bhuiyan, M., Krishna, A., & Ghose, A. (2006a). "A Combined Approach for Supporting the Business Process Model Lifecycle," *in Proceedings of the 10th Pacific Asia Conference on Information Systems (PACIS)*, Kuala Lumpur, Malaysia: AIS Electronic Library, July 6-9

Koliadis, G., Vranesevic, A., Bhuiyan, M., Krishna, A., & Ghose, A. (2006b). "Combining i* and BPMN for Business Process Model Lifecycle Management," *in Proceedings of the Business Process Management Workshops*, Vienna, Austria: Springer, Sep 4-7, pp. 416-427

Kongdenfha, W., Motahari-Nezhad, H. R., Benatallah, B., Casati, F., & Saint-Paul, R. (2009). "Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters," *IEEE Transactions on Services Computing*, vol. 2 no. 2, pp. 94-107.

Kongdenfha, W., Saint-Paul, R., Benatallah, B., & Casati, F. (2006). "An Aspect-Oriented Framework for Service Adaptation," *in Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC)*, Chicago, IL, USA: ACM Press, Nov 15-19, pp. 15–26

Kowalski, R. & Sergot, M. (1986). "A Logic-Based Calculus of Events," *New Genereation Computing*, vol. 4 no. 1, pp. 67-95.

Koymans, R. (1992). *Specifying Message Passing and Time-Critical Systems with Temporal Logic*: Springer-Verlag.

Kramer, J. & Magee, J. (2007). "Self-Managed Systems: an Architectural Challenge," in *2007 Future of Software Engineering (FOSE '07)*: IEEE Computer Society, pp. 259-268.

Krishna, A., Ghose, A., & Vilkomir, S. (2004). "Co-Evolution of Complementary Formal and Informal Requirements," *in Proceedings of the 7th International Workshop Principles of Software Evolution (IWPSE 2004)*, Kyoto, Japan: IEEE Computer Society, Sep 6-7, pp. 159-164

Krishna, A., Vilkomir, S., & Ghose, A. (2009). "Consistency-Preserving Co-evolution of Formal Specifications and Agent-Oriented Conceptual Models," *Information and Software Technology*, vol. 51 no. 2, pp. 478-496.

Kruchten, P. (1995). "The 4+1 view of architecture," *IEEE Software*, vol. 12 no. 6 (Nov), pp. 42-50.

Lago, P., Avgeriou, P., & Hillard, R. (2010). "Software Architecture: Framing Stakeholders' Concerns (Guest Editors' Introduction)," *IEEE Software*, vol. 27 no. 6 (Nov), pp. 20-24.

Lamanna, D., Skene, J., & Emmerich, W. (2003). "SLAng: A Language for Service Level Agreements.," *in Proceedings of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS 2003)*, San Juan, Puerto Rico: IEEE Computer Society Press, May 28-30, pp. 100-106

Lamsweerde, A. v. (2003). "Proceedings of From System Goals to Software Architecture," in *Formal Methods for Software Architectures*, vol. 2804, *LNCS*, M. B. P. Inverardi, Ed. Bertinoro, Italy: Springer-Verlag, pp. 25-43.

Lamsweerde, A. v. (2004). "Goal-Oriented Requirements Enginering: A Roundtrip from Research to Practice," *in Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, Kyoto, Japan: IEEE Computer Society, September 6-10, pp. 4-7

Larman, C. & Basili, V. R. (2003). "Iterative and Incremental Development: A Brief History," *IEEE Computer*, vol. 36 no. 6, pp. 47-56.

Lau, D. & Mylopoulos, J. (2004). "Designing Web Services with Tropos," *in Proceedings of the Second International Conference on Web Services (ICWS'04)*, San Diego, CA, USA: IEEE Computer Society, July 6-9, 2004, pp. 306-313

Letier, E., Kramer, J., Magee, J., & Uchitel, S. (2005). "Fluent Temporal Logic for Discrete-Time Event-Based Models," *in Proceedings of the 10th European Software Engineering Conference (ESEC) held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FOSE)*, Lisbon, Portugal: ACM, Sep 5-9, pp. 70-79

Letier, E., Kramer, J., Magee, J., & Uchitel, S. (2008). "Deriving Event-Based Transition Systems from Goal-Oriented Requirements Models," *Automated Software Engineering*, vol. 15 no. 2 (June), pp. 175-206.

Lewerentz, C. & Lindner, T., Eds. (1995). *Formal Development of Reactive Systems - Case Study Production Cell*, LNCS vol. 891: Springer-Verlag.

Li, Z., Han, J., & Jin, Y. (2005). "Pattern-Based Specification and Validation of Web Services Interaction Properties," *in Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands: Springer, Dec 12-15, pp. 73 - 86

Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., & Mylopoulos, J. (2006). "On Goal-based Variability Acquisition and Analysis," *in Proceedings of the 14th IEEE International Requirements Engineering Conference*, Minneapolis/St.Paul, Minnesota, USA: IEEE Computer Society, Sep 11-15, pp. 76-85

Lopes, A., Fiadeiro, J. L., & Wermelinger, M. (2001). "A Compositional Approach to Connector Construction," *in Proceedings of the 15th Workshop on Algebraic Development Techniques (WADT)*, Genova, Italy: Springer, April 1-3, pp. 201-220

Ludwig, H., Keller, A., Dan, A., King, R. P., & Franck, R. (2003). "Web Service Level Agreement (WSLA) Language Specification Version 1.0," IBM Corporation.

Mahfouz, A., Barroca, L., Laney, R., & Nuseibeh, B. (2009). "Requirements-Driven Collaborative Choreography Customization," *in Proceedings of the International Conference on Service-Oriented Computing (ICSOC'09)*, Stockholm, Sweden: Springer, Nov 24-27, pp. 144-158

Mallya, A. U. & Singh, M. P. (2006). "Incorporating Commitment Protocols into Tropos," in *Agent-Oriented Software Engineering VI*, vol. 3950, *Lecture Notes in Computer Science*: Springer, pp. 69-80.

Martens, A. (2005). "Analyzing Web Service Based Business Processes," *in Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE'05), part of the 2005 European Joint Conferences on Theory and Practice of Software (ETAPS'05)*, Edinburgh, Scotland: Springer-Verlag, April 4-8

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., et al. (2004). "OWL-S: Semantic Markup for Web Services - Techincal Overview," DAML-S Coalition.

Martin-Flatin, J.-P., Sventek, J., & Geihs, K. (2006). "Introduction to Special Issue on Self-Managed Systems and Services," *Communications of the ACM*, vol. 49 (2006), pp. 36-39.

Mazzoleni, P. & Srivastava, B. (2008). "Business Driven SOA Customization," *in Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC)*, Sydney, Australia: Springer-Verlag, Dec 1-5, pp. 286 - 301

Mellor, S. J., Scott, K., Uhl, A., & Weise, D. (2004). *MDA Distilled: Principles of Model-Driven Architecture*: Addison-Wesley.

Mendling, J. & Hafner, M. (2005). "From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL," *in Proceedings of the On the Move to Meaningful Internet Systems (OTM Confederated International Conferences) OTM Workshops*, Agia Napa, Cyprus: Springer, Oct 31-Nov 4, pp. 506-515

Mens, T. & D'Hondt, T. (2000). "Automating Support for Software Evolution in UML," *Automated Software Engineering*, vol. 7 no. 1, pp. 39-59.

Meyer, B. (1997). *Object-Oriented Software Construction, Second Edition*: Prentice Hall.

Milner, R. (1999). *Communicating and Mobile Systems: the Pi-Calculus.*: Cambridge University Press.

Nezhad, H. R. M., Benatallah, B., Martens, A., Curbera, F., & Casati, F. (2007). "Semi-Automated Adaptation of Service Interactions," *in Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, Banff, Alberta, Canada: ACM, May 8-12, pp. 993-1002

Nitto, E. D., Ghezzi, C., Metzger, A., Papazoglou, M., & Pohl, K. (2008). "A Journey to Highly Dynamic, Self-Adaptive Service-Based Applications," *Automated Software Engineering*, vol. 15 no. 3-4 (December), pp. 313-341.

Nuseibeh, B. & Easterbrook, S. (2000). "Requirements Engineering : A Roadmap," *in Proceedings of the Conference on The Future of Software Engineering (FOSE) in Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, Limerick, Ireland: ACM Press, June 4-11, pp. 35-46

Nuseibeh, B., Kramer, J., & Finkelstein, A. (1994). "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification," *IEEE Transactions Software Engineering*, vol. 20 no. 10, pp. 760-773.

OASIS (2009). "Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2," OASIS Web Services Reliable Exchange (WS-RX) TC.

O'Leary, D. (2009). "The Most Cited IEEE Software Articles," *IEEE Software*, vol. 26 no. 1 (January), pp. 12-14.

OMG-MOF (2010). "Meta Object Facility (MOF) Core Specification Version 4.2," Object Management Group ptc/2010-12-08, December.

OMG-OCL (2010). "Object Constraint Language (OCL) v2.3 i," Object Management Group ptc/2010-11-42, December.

OMG-QVT (2002). "MOF 2.0 Query / Views / Transformations RFP," Object Management Group ad/2002-04-10.

OMG-SOAML (2009). "Service Oriented Architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)," Object Management Group ptc/2009-12-09, Dec 9.

OMG-UML (2004). "Unified Modeling Language (UML) Specification: Infrastructure," Object Management Group (OMG) ptc/04-10-14, November.

Orriëns, B. & Yang, J. (2006). "A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaborations," *in Proceedings of the IEEE International Conference on Services Computing (SCC)*, Chicago, Illinois, USA: IEEE Computer Society, Sep 18-22, pp. 182-189

Orriëns, B., Yang, J., & Papazoglou, M. (2003a). "A Framework for Business Rule Driven Service Composition," *in Proceedings of the 4th International Workshop on Technologies for E-Services (TES 2003)*, Berlin, Germany: Springer-Verlag, Sep 8, pp. 14-27

Orriëns, B., Yang, J., & Papazoglou, M. (2003b). "Model Driven Service Composition," *in Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento - Italy, Dec 15-18, pp. 75-90

Osborne, M. J. & Rubinstein, A. (1994). *A Course in Game Theory*: MIT Press.

Padgham, L., Thangarajah, J., & Winikoff, M. (2005). "Tool Support for Agent Development using the Prometheus Methodology," *in Proceedings of the 5th International Conference on Quality Software (QSIC'05)*, Melbourne, Australia: IEEE Computer Society, Sep 19-20, pp. 383 - 388

Padgham, L. & Winikoff, M. (2004). *Developing Intelligent Agent Aystems: A Practical Guide*: John Wiley & Sons.

Padgham, L., Winikoff, M., DeLoach, S., & Cossentino, M. (2008). "A Unified Graphical Notation for AOSE," *in Proceedings of the 9th International Workshop on Agent-Oriented Software Engineering (AOSE 2008)*, Estoril, Portugal: Springer-Verlag, May 12-13, pp. 116-130

Papazoglou, M. (2008). "The Challenges of Service Evolution," *in Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, Montpellier, France: Springer-Verlag, June 18-20, pp. 1-15

Papazoglou, M. & Georgakopoulos, D. (2003). "Service-Oriented Computing," *Communications of the ACM*, vol. 46 no. 10 (October), pp. 25-28.

Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F., & Krämer, B. J. (2005). "Service-Oriented Computing: A Research Roadmap," *in Proceedings of the Dagstuhl Seminar*

Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F., & Krämer, B. J. (2006). "Service-Oriented Computing: A Research Roadmap," *in Proceedings of the Dagstuhl Seminar on Service Oriented Computing*: IBFI, May 5

Parnas, D. L. (1972). "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15 no. 12 (December).

Peltz, C. (2003). "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36 no. 10 (October), pp. 46-52.

Penserini, L., Perini, A., Susi, A., & Mylopoulos, J. (2006). "From Stakeholder Needs to Service Requirements," in *Proceedings of the Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'06) co-located with the 14th IEEE International Requirements Engineering Conference (RE'2006)*. Minneapolis, Minnesota, USA: IEEE Computer Society, pp. 8-17.

Penserini, L., Pernini, A., Susi, A., & Mylopoulos, J. (2007). "High Variability Design for Software Agents: Extending Tropos," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 2 no. 4 (November).

Pesic, M. & van der Aalst, W. (2006). "A Declarative Approach for Flexible Business Processes Management," *in Proceedings of the Business Process Management Workshops*, Vienna, Austria: Springer, Sep 4-7, pp. 169-180

Petri, C. A. (1962). "Kommunikation mit Automaten," PhD thesis, Institut f¨ur Instrumentelle Mathematik, Bonn, Germany.

Ponnekanti, S. R. & Fox, A. (2002). "SWORD: A Developer Toolkit for Web Service Composition," *in Proceedings of the Eleventh World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, USA, May 7-11

Potter, S., Eds. (2006). *Doing Postgraduate Research*, 2nd ed. London: Sage Publications.

Prieto-Diaz, R. (1990). "Domain Analysis: An Introduction," *SIGSOFT Software Engineering Notes,*, vol. 15 no. 2, pp. 47–54.

Quartel, D., Dijkman, R., & van Sinderen, M. (2004). "Methodological Support for Service-Oriented Design with ISDL," *in Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, New York, NY, USA: ACM, Nov 15-19, pp. 1-10

Quartel, D., Dijkman, R., & van Sinderen, M. (2005). "An Approach to Relate Business and Application Services Using ISDL," *in Proceedings of the 9th IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, Enschede, The Netherlands: IEEE Computer Society, Sep19-23, pp. 157-168

Quartel, D., Pires, L. F., & van Sinderen, M. (2002). "On Architectural Support For Behaviour Refinement In Distributed Systems Design," *Journal of Integrated Design & Process Science*, vol. 6 no. 1, pp. 1-30.

Rao, J. & Su, X. (2004). "A Survey of Automated Web Service Composition Methods," *in Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, San Diego, California, USA: Springer, July 6, pp. 43-54

Reichert, M., Rinderle-Ma, S., & Dadam, P. (2009). "Flexibility in Process-Aware Information Systems," in *Transactions on Petri Nets and Other Models of Concurrency II*, vol. 5460, *LNCS*: Springer, pp. 115-135.

Rinderle, S., Reichert, M., & Dadam, P. (2004). "Correctness Criteria for Dynamic Changes in Workflow Systems: A Survey," *Data & Knowledge Engineering - Special Issue on Advances in Business Process Management*, vol. 50 no. 1, pp. 9-34.

Rinderle, S., Wombacher, A., & Reichert, M. (2006). "On the Controlled Evolution of Process Choreographies," *in Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, Georgia, USA: IEEE Computer Society, April 3-8, pp. 124-124

Ross, D. T. & Schoman, K. E. (1977). "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, vol. 3 no. 1, pp. 6-15.

Ross-Talbot, S. & Fletcher, T. (2006). "Web Services Choreography Description Language: Primer," W3C.

Singh, M. P. (1999). "An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts," *Artificial Intelligence and Law*, vol. 7 no. 1, pp. 97-113.

Singh, M. P., Chopra, A. K., & Desai, N. (2009). "Commitment-Based Service-Oriented Architecture," *IEEE Computer*, vol. 42 no. 11 (November), pp. 72-79.

Skene, J. (2007). "Language Support for Service-Level Agreements for Application-Service Provision," Department of Computer Science, University College London, London.

Skene, J., Lamanna, D., & Emmerich, W. (2004). "Precise Service Level Agreements," *in Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, Edinburgh, Scotland: IEEE Computer Society Press, pp. 179-188

Skene, J., Piccinelli, G., & Stearns, M. (2003). "Modelling Electronic Service Systems Using UML," *in Proceedings of the Workshop on Service-Based Software Engineering (FME'03/SBSE)*, Pisa, Italy, Sep 8-9, pp. 15-30

Skogan, D., Grønmo, R., & Solheim, I. (2004). "Web Service Composition in UML.," *in Proceedings of the 8th International IEEE Enterprise Distributed Object Computing (EDOC)*, Monterey, California, USA: IEEE Computer Society, Sep 20-24, pp. 47-57

Sommerville, I. & Sawyer, P. (1997). *Requirements Engineering: A Good Practice Guide*: Wiley.

Su, J., Bultan, T., Fu, X., & Zhao, X. (2008). "Towards a theory of web service choreographies," in *Proceedings of the 4th international conference on Web services and formal methods*. Brisbane, Australia: Springer-Verlag, pp. 1-16.

Susi, A., Perini, A., Mylopoulos, J., & Giorgini, P. (2005). "The Tropos Metamodel and its Use," *Informatica*, vol. 29 no. 4, pp. 401-408.

Swanson, E. B. (1976). "The Dimensions of <aintenance," *in Proceedings of the 2nd international conference on Software engineering (ICSE '76)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct 13-15, pp. 492-497

Swanson, E. B. & Lientz, B. P. (1980). *Software Maintenance Management*. Reading, MA: Addison-Wesley.

Tan, Y.-H. & Thoen, W. (1998). "A Logical Model of Directed Obligations and Permissions to Support Electronic Contracting," *International Journal of Electronic Commerce*, vol. 3 no. 2, pp. 87-104.

Telang, P. R. & Singh, M. (2010). "Abstracting and Applying Business Modeling Patterns from RosettaNet," *in Proceedings of the 8th International Conference on Service-Oriented Computing*, San Francisco, California, USA: Springer, Dec 7-10, pp. 426-440

Telang, P. R. & Singh, M. P. (2009). "Enhancing Tropos with Commitments: A Business Metamodel and Methodology," in *Conceptual Modeling: Foundations and Applications*, vol. 5600, *LNCS*, A. Borgida, V. Chaudhri, P. Giorgini, & E. Yu, Eds.: Springer, pp. 417-435.

Thom, L. H., Reichert, M., & Iochpe, C. (2009). "Activity Patterns in Process-aware Information Systems: Basic Concepts and Empirical Evidence," *International Journal of Business Process Integration and Management*, vol. 4 no. 2, pp. 93-110.

Trastour, D., Preist, C., & Coleman, D. (2003). "Using Semantic Web Technology to Enhance Current Business-to-Business Integration Approaches," *in Proceedings of the 7th International Enterprise Distributed Object Computing Conference (EDOC 2003)*, Brisbane, Australia: IEEE Computer Society, Sep 16-19, pp. 222-230

Traverso, P., Pistore, M., Roveri, M., Marconi, A., Kazhamiakin, R., Lucchese, P., et al. (2004). "Supporting the Negotiation between Global and Local Business Requirements in Service Oriented Development," ITC-IRST, Trento, Italy Technical Report T04-08-03.

Treaster, M. (2005). "A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems," *ACM Computing Research Repository (CoRR)*, vol. 501002, pp. 1-11.

van der Aalst, W. (2003). "Don't go with the flow: Web Services composition standards exposed," *IEEE Intelligent Systems*, vol. 18 no. 1, pp. 72-76.

van der Aalst, W. (2004). "Inheritance of Interorganizational Workflows: How to agree to disagree without losing control?," *Information Technology and Management*, vol. 4 no. 4 (October), pp. 345-389.

van der Aalst, W. & Pesic, M. (2006). "DecSerFlow: Towards a Truly Declarative Service Flow Language," *in Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM'06) co-located with the 4th International Conference on Business Process Management (BPM 2006)*, Vienna, Austria: Springer, Sep 12, pp. 1-23

van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., & Barros, A. (2003a). "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14 no. 3, pp. 5–51.

van der Aalst, W., van Dongen, B. F., Herbst, J., Maruster, L., G.Schimm, & Weijters, A. J. M. M. (2003b). "Workflow Mining a Survey of Issues and Approaches," *Data and Knowledge Engineering*, vol. 47 no. 2, pp. 237-267.

van der Aalst, W. & Weijters, A. J. M. M. (2005). "Process Mining," in *Process-Aware Information Systems: Bridging People and Software through Process Technology*: Wiley & Sons.

van Lamsweerde, A. (2000). "Requirements Engineering in the Year 00: A Research Perspective (Keynote paper)," *in Proceedings of the 22nd International Conference on Software Engineering (ICSE'2000)*, Limerick, Ireland: IEEE Press, June 4-11

van Lamsweerde, A. (2001). "Goal-Oriented Requirements Engineering: A Guided Tour," *in Proceedings of the 5th International Symposium on Requirements Engineering (RE'01)*, Toronto, Canada: IEEE Computer Society Press, Aug 27-31, pp. 249-261

van Lamsweerde, A., Dardenne, A., Delcourt, B., & Dubisy, F. (1991). "The KAOS Project: Knowledge Acquisition in Automated Specification of Software," *in Proceedings of the AAAI Spring Symposium Series, Design of Composite Systems Track*, Stanford University, Stanford, California, USA, March, pp. 59-62

Vasconcelos, A., Caetano, A., Neves, J., Sinogas, P., Mendes, R., & Tribolet, J. M. (2001). "A Framework for Modeling Strategy, Business Processes and Information Systems," *in Proceedings of the 5th International Enterprise Distributed Object Computing Conference (EDOC 2001)*. Seattle, Washington, USA: IEEE Computer Society, Sep 4-7, pp. 69-80

Vilet, H. V. (2001). *Software Engineering: Principles and Practice*, 2nd ed: John Wiley & Sons, Inc.

W3C-SOAP (2007). "Simple Object Access Protocol (SOAP) Version 1.2," W3C XML Protocol Working Group, April 27.

W3C-XSL (2007). "XSL Transformations (XSLT) Version 2.0," W3C, XSL Working Group.

Weber, B., Rinderle, S., & Reichert, M. (2007). "Change Patterns and Change Support Features in Process-Aware Information Systems," *in Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway: Springer, June 11-15, pp. 574-588

White, S. A. (2004). "Business Process Modeling Notation (BPMN) Version 1.0," Business Process Management Initiative (BPMI.org), May.

Wieczorek, S., Roth, A., Stefanescu, A., Kozyura, V., Charfi, A., Kraft, F. M., et al. (2009). "Viewpoints for Modeling Choreographies in Service-Oriented Architectures," *in Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture 2009 and European Conference on Software Architecture (WICSA/ECSA)*, Cambridge, UK: IEEE, Sep 14-17, pp. 11-20

Wing, J. M. (1988). "A Study of 12 Specifications of the Library Problem," *IEEE Software*, vol. 5 no. 4 (July), pp. 66-76.

Winikoff, M. (2009). "Future Directions for Agent-Based Software Engineering," *International Journal of Agent-Oriented Software Engineering*, vol. 3 no. 4, pp. 402-410.

Wombacher, A. (2009). "Alignment of Choreography Changes in BPEL Processes," *in Proceedings of the IEEE International Conference on Services Computing (SCC'09)*, Bangalore, India: IEEE, Sep 21-25, pp. 1-8

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*: John Wiley & Sons.

Wooldridge, M. & Jennings, N. R. (1995). "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, vol. 10 no. 2, pp. 115-152.

Xu, L., "A Framework for E-markets: Monitoring Contract Fulfillment," in *Web Services, E-Business, and the Semantic Web*, vol. 3095, *Lecture Notes in Computer Science*, C. Bussler, D. Fensel, M. Orlowska, & J. Yang, Eds.: Springer Berlin / Heidelberg, 2004, pp. 51-61.

Yin, R. (1994). *Case Study Research: Design and Methods*, 2nd ed: Sage Publications (CA).

Yu, E. (1993). "Modelling Organizations for Information Systems Requirements Engineering," *in Proceedings of the 1st International Symposium on Requirements Engineering (RE'93)*, San Diego, California, USA, Jan 4-6, pp. 34-41

Yu, E. (1995). *Modelling Strategic Relationships for Process Reengineering*. Toronto, Canada: PhD thesis, Department of Computer Science, University of Toronto.

Yu, E. (1997). "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering," *in Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, Washington DC, USA, Jan 6-10, pp. 226-235

Yu, E. & Mylopoulos, J. (1994). "Understanding "Why" in Software Process Modelling, Analysis, and Design," *in Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*, Sorrento, Italy: IEEE Computer Society, May 16-21, pp. 159-168

Zaha, J. M., Barros, A., Dumas, M., & ter Hofstede, A. (2006a). "Let's Dance: A Language for Service Behavior Modeling," in *On the Move to Meaningful Internet Systems (OTM Confederated International Conferences) OTM Workshops*, vol. 4275/2006, *Lecture Notes in Computer Science*: Springer Berlin / Heidelberg, pp. 145-162.

Zaha, J. M., Dumas, M., ter Hofstede, A., Barros, A., & Decker, G. (2006b). "Service Interaction Modeling: Bridging Global and Local Views," *in Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006)*, Hong Kong, China: IEEE Computer Society, 16-20 October, pp. 45-55

Zaha, J. M., Dumas, M., ter Hofstede, A., Barros, A., & Decker, G. (2008). "Bridging Global and Local Models of Service-oriented Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 38 no. 3 (May), pp. 302-318.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). "Developing Multiagent Systems: The Gaia Methodology," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 12 no. 3 (July), pp. 317-370.

Zave, P. (1997). "Classification of Research Efforts in Requirements Engineering," *ACM Computing Surveys*, vol. 29 no. 4, pp. 315-321.

Zave, P. & Jackson, M. (1997). "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6 no. 1, pp. 1-30.

Zhao, X., Yang, H., & Qiu, Z. (2006). "Towards the Formal Model and Verification of Web Service Choreography Description Language," *in Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM'06) co-located with the 4th International Conference on Business Process Management (BPM 2006)*, Vienna, Austria: Springer, Sep 8-9, pp. 273-287

Owner:	GPS013
File Name:	p_552387_1341916374_20120710a:
Date:	July 11, 2012 10:02:02 a.m.
Hostname:	dm03

Copies:	1
Images Processed:	201
Total Images:	201

Operator Message:

Printed Sides:	2–Sided
Collate:	On
Stack:	Offset
Staple:	None
Fold:	None
Document Exit:	Stapler
Format/PDL:	PostScript®

```
%%[ Warning: ^D characters ignored ]%%
%!PS-Adobe-3.0
%%Pages: 201
%KDKJobId: S0165673
%KDKSource: remove file:/var/spool/kodak_resources/rip_data/hch.27741.721847
%!PS-Adobe-3.0
%%Driver Build: 08.00.0015
%%StartKPDComments
%EK-JSE-3-0-Build: 08.01.0004
%KDKBody: (A4Punched) on
%KDKPrintMethod: print
%KDKOutputMedia:  stapler
%%Requirements:  numcopies(1) staple(none)  fold(none)  duplex(on)  collate jog(alternat
%KDKRequirements: numcopies(1) staple(none)  fold(none)  duplex(longedge)  collate jog(a
%KDKCovers: (A4Punched) none
%KDKRotation: 0
%KDKError: on (A4Punched)
%KDKActOn:
%%Title: (p_552387_1341916374_20120710amahfouzdissertationfinalfinalfinalprinta4.pdf)
%%For: (GPS013)
%%Emulation: postscript
%%Routing: ()
%%DocumentPrinterRequired: () (DS9110)
%%DocumentMedia: (A4Punched) 595 842 80 (white) (punched)
%KDKMedia: (A4Punched) sides(2)
%KDKEngineGRET: off
%KDKSpool: on
%KDKReqPunch: 0
%KDKReqReorderSheets: facedown
%%EndKPDComments
```