Conceptual Modelling of Adaptive Web Services based on High-level Petri Nets

PhD Thesis

BASSAM ZAFAR

Software Technology Research Laboratory

Faculty of Computing Sciences and Engineering

De Montfort University

2008

Declaration

I declare that the work described in my thesis is original work undertaken by me between December 2004 to September 2008 for the degree of Doctor of Philosophy, at De Montfort University, United Kingdom. This thesis is written by me and produced using LATEX.

Acknowledgements

This thesis is dedicated to my family, friends and colleagues whose support has proved invaluable to my research. Most importantly, I would like to express my sincere thanks to my supervisor, Professor Hussein Zedan , without his support, encouragement and guidance this thesis would have been an impossible undertaking. I wish to thank Dr. Monika Solanki, who helped me in getting started in the initial years of my research and provided helpful suggestions during many varied discussions.

Also, I wish to express my sincere and deepest gratitude to my parents, without whose love, care and support, it would have been very difficult for me to achieve my goals. My father made sure that I was never bothered with any responsibilities other than my research and for which I would like to express my heartfelt thanks. My parents have been a constant source of moral and financial support for most part of my life. They gave me the hope and support that made this work possible. If this thesis were worth the time spent away from them, each letter on every page would be priceless. For their love, generosity, and understanding, my gratitude is inexpressible. I could not possibly give back to my parents what they have lost in their efforts to support me - I try to honour their sacrifice. I have dedicated this thesis to them with love and respect.

I would like to express my deepest love and gratitude for my wife, who stood by me like nobody else in all these difficult years and has offered me her constant support,patience, encouragement, unconditional love and life. I dedicate this thesis to her too, with all my heart.

Last but not least I would like to thank all my colleagues at the Software Technology Research Laboratory - De Montfort University for their constant support and for making STRL such a stimulating and friendly place for research.

Publications

[1] Bassam Zafar, Hussein Zedan, Monika Solanki. Towards A Service-Driven Petri Nets-Based Conceptual Model for Web services. IADIS International Conference e-Society, volume 2, pages 338–343, 2006.

Abstract

Service technology geared by its SOA architecture and enabling Web services is rapidly gaining in maturity and acceptance. Consequently, most worldwide (private and corporate) cross-organizations are embracing this paradigm by publishing, requesting and composing their businesses and applications in the form of (web-)services. Nevertheless, to face harsh competitiveness such service-oriented cross-organizational applications are increasingly pressed to be highly composite, adaptive, knowledge-intensive and very reliable. In contrast to that, Web service standards such as WSDL, WSBPEL , WS-CDL and many others offer just static, manual, purely process-centric and ad-hoc techniques to deploy such services.

The main objective of this thesis consists therefore in leveraging the development of service-driven applications towards more reliability, dynamically and adaptable knowledge-intensiveness. This thesis puts forward an innovative framework based on distributed high-level Petri nets and event-driven business rules. More precisely, we developed a new variant of high-level Petri Nets formalism called Service-based Petri nets (CSrv-Nets), that exhibits the following potential characteristics. Firstly, the framework is supported by a stepwise methodology that starts with diagrammatical UML-class diagrams and business rules and leads to dynamically adaptive services specifications. Secondly, the

framework soundly integrates behavioural event-driven business rules and stateful services both at the type and instance level and with an inherent distribution. Thirdly, the framework intrinsically permits validation through guided graphical animation. Fourthly, the framework explicitly separates between orchestration for modelling rule-intensive single services and choreography for cooperating several services through their governing interactive business rules. Fifthly, the framework is based on a two-level conceptualization: (1) the modelling of any rule-centric service with CSrv-Nets; (2) the smooth upgrading of this service modelling with an adaptability-level that allows for dynamically shifting up and down any rule-centric behavior of the running business activities.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
BPEL4WS	Business Process Execution Language For Web Service
CORBA	Common Object Requesting Broker Architecture
DAML-S	Darpa Agent Markup Language for Services
DCOM	Distributed Component Object Model
ECA	Event Condition Action
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
HTTPR	Reliable Hyper Text Transfer Protocol
J2EE	Java Platform, Enterprise Edition
MIME	Multipurpose Internet Mail Extensions
OCL	Object Constraint Language
OWL-S	Web Ontology Language for Services
RPC	Remote Procedure Call

RuleML	Rule Markup Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SOA	Service Oreiented Architecture
SOC	Service Oriented Computing
UDDI	Universal Description Discovery and Integration
UML	Unified/Universal Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSBPEL	Web Service Business Process Execution Language (was earlier BPEL4WS)
WS-CDL	Web Service Choreography Description Language
WSCI	Web Service Choreography Interface
WSDL	Web Service Description Language
WSFL	Web Service Flow Language
WSMO	Web Service Modeling Ontology
WWW	World Wide Web
XML	Extensible Markup Language

Glossary

The following glossary terms are taken from the Open Distributed Processing Reference Model [40], and various Web service specification including: [33, 28, 56].

Term Applied To Web service	Definition
Interface	A service interface is the abstract boundary that a service express. It defines the types of messages and the message exchange patterns that are involved in interacting with the service.
Problem Domain	The functional area of interest , or under control, by individual or groups of (web) services hosted on the interest and accessible either locally or globally (to other service groups) to fulfill a task or a series of tasks within a function area
Composition	A web service composition consists of an orchestration of web service interactions defined in a local process (itself potentially a service). Static web service compositions are those which use services known at design time and are bound to a composition at design time. Dynamic web service compositions those which define web service interactions where the services are not known at design time, and which are discovered or their properties resolved based upon a criteria process set at design time.
Orchestration	Describes the definition and the implementation of processes that drives the message exchanges between one or more web services. The BPEL4WS standard refers to participating services as composition Partners. Interaction is seen between one process and many services (i.e. one to many).
Choreography	Choreography describes the collective message exchange among interacting Web Services, providing a global, message-oriented view of the interactions (observing and controlling a many to many relationship)
Service	A web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols

Chapter 1

Introduction

The objectives of this introductory chapter are fourfold. Firstly, this chapter motivates the general context and research scope of this thesis within the service paradigm field, and then motivate the topical research challenges we have been focussing on. Secondly, we go into detail about the main research questions tackled in this thesis and its ramifications, as driving forces in reshaping the thesis contributions. Thirdly, in accordance with set up research questions and objectives, we enumerate the main original contributions of this thesis. Finally, this chapter highlights the content of the remaining chapters.

1.1 Motivation and Research Scope

Over recent years there were an increasingly dominating market globalisation, geared by highly unpredictable volatility and fierce competition. In parallel, the advances and confluence of computation and wireless communications have been boosting the omnipresence and pervasiveness of the internet and the World Wide Web (WWW) [11], anywhere, anytime and through any communication-aware devices and channels. As a consequence, on the one hand, to stay competitive most organisations (public and private) are intensively collaborating their know-how by dynamically building loosely-coupled networked cross-organisational giants. On the other hand, the internet has been leveraged from just a glue of (syntactical) information to an unavoidable complex scene for networking and composing such as cross-organisational, knowledge-intensive and interaction-centric cooperations; mainly enabled through the so-called service-oriented architectures and Web services.

Service-oriented computing (SOC) presently represents one of the emerging technological innovations in accurately (semi-)automating these new business requirements [33]. Indeed, as a new computing paradigm, this technology treats distribution, loose-coupling and heterogeneity as driving first class principles and mechanisms. Service technology principles are thus centered on the unlimited capabilities and pervasiveness of internet technologies and the World Wide Web.

Web services enabled by service-oriented architectures (SOA) in this technology are platform-independent, self-contained software entities with explicit interfaces, adequately instantiate to be described, published, discovered and deployed on the Web. Besides that is undoubtedly the capability of Web services to be composed with other basic or composite Web services to form large scale evolvable business applications, that dynamically fulfil more realistic requirements of private or public requesters (i.e. P2B and B2B). As a service technology, Web services can thus be manipulated (e.g. described, published, discovered and composed) using adequate standards described in terms of XML-based languages, such as WSDL [7], UDDI [92], WSBPEL [87] and WSCI [38]. These standardised languages have been rapidly gaining in maturity and becoming widely adopted, thereby increasingly attracting worldwide cross-organisations to embrace this service technology for (semi)-automating their business and inherent networked information systems.

This significant technological shift towards the wide adoption of service-oriented architecture and computing enable Web services at this large organisational scale at a rapid pace. Web services have been pressing for more ad-hoc deployment techniques to cope with an ever growing demand on complex yet realistic composite services. Beyond the originally process-centric, rigid and static compositions of services using mainly the orchestration-driven WSBPEL and partially the global choreography-driven WS-CDL [1] and WSCI. Therefore, there is a need for more ad-hoc technology-driven development techniques of non-traditional advanced services characterised in particular by their:

Persistency and Conversation: Complex realistic service-driven business applications (i.e. applications aimed to be deployed using service standards and architectures), are required to be conversational and highly persistent as they are mostly long-lived term systems. Tackling persistency means providing advanced abstraction mechanisms , such as those provided by the Object paradigm for example classification, inheritance and roles around the de facto standard UML method [32].

Unfortunately, even at the technological level no contemporary composition standard is able to intrinsically address persistency and conversation. The widely adopted WSBPEL, for instance, uses very restricted data variables that automatically vanish after execution.

Knowledge-intensiveness: Service-driven applications such as E-commerce, E-banking, E-Government or E-health are overwhelmingly governed by huge knowledge described mostly in terms of business rules regulating how to do business at the intra- and cross-organisational levels alike. Restricting business activities to just exchanging messages as intended in Web standards like WSDL and WSBPEL, thus represents serious obstacles to cope with the inherent rich knowledge encompassing such activities. Although, several ad-hoc deployment techniques are starting to emerge ranging from XML rule-based languages such as reactive RuleML [91, 23], DAML-S [5], OWL-S [60] or aspect-oriented techniques [19, 51, 50]; the handling of such knowledge is still not satisfactory.

Required Complementary local/global Composition: As a realistic service application is usually composed of numerous interacting services, both service-focussed and global inter-service are deemed necessary. Unfortunately, with the popularity of WSBPEL composition-like language, the single service-focussed orchestration remains the only dominating perspective, whereas the global inter-service interactions among composite Web services, the so-called choreography is getting less attention or treated independently from the orchestration vision. We claim that harmoniously addressing both local and global visions, while composing services, in a complementary and uniform manner represents an important step towards more realistic composite services.

High Distribution: Most current standards for service composition do not cope with decentralised architectures, where different business process activities can reside in different locations. As benefits of distribution we may cite, on the one hand, the intrinsic support of concurrent activities within a business process.On the other hand, distribution enhances the migrate of services and their activities in accordance with the target (user's) location and computational resources.

Dynamic Adaptivity and Evolution: To stay competitive, today's services must cater for high flexibility and adaptability as business markets dynamically and rapidly change and opportunistic alliances are highly favored. Particularly composite services are required to be highly and dynamically adaptable to cope with different variants of requesters and their evolving requirements. Moreover, when the composition of services is flexible and dynamically adaptable, the number of application (cross-organisation) business processes significantly decrease as they become mostly reusable. Consequently, development efforts, costs and (process) size-complexity are being more mastered.

The direct technological ad-hoc realisation of such intractable advanced services is unfortunately not without serious pitfalls, limitations, unnecessary costly and risky investments by organisations acquiring them. Indeed, even without referring to such advanced services, the promise of service technology in delivering by its own adaptive composite process-centric services is still a far-off objective. Adding such multi-concern requirements (e.g. knowledge-intensiveness, distribution and harmonious local/global composition) just means more inflexible and spaghetti-like hard-coded services, impossible to build let alone to compose and adapt. Moreover, the difficulties of uniformly and coherently addressing the afore-mentioned service advanced characteristics, has resulted in deployment architecture focussing mostly on one or two issues while ignoring the others.

In response to this unsatisfactory state-of-affairs, we are thus witnessing a strong consensus that this technology must be embolden and steered by prior, deep understanding and handling of most advanced service requirements at the business and foundational levels, and only afterwards addressing a deployment to accurately and progressively mirror that domain-level understanding, while taking benefits and reshaping available service technology as a consequence. Indeed, most of the above crucial service requirements such as multi-concern knowledge and adaptability as well as their explicit and balanced separation of concerns are excellent domain-level issues.

Instead of vainly attempting to disparately enforce these advanced service features through syntactical "codification" and thus loosing their essence, we endeavour in this thesis first to faithfully elicit, understand and validate them at the business-foundation levels, where all stakeholders (e.g. managers, analysts, developers, users and finally programmers) may take a lions share in these activities, and where tailored semi- and formal techniques ensure a high-level of flexibility and reliability (through validation and verification).

Therefore, this thesis objective concerns the leveraging of the service paradigm from its dominating technology-dependency towards more stepwise service engineering life-cycle development, where early phases of business requirements such as elicitation, modelling and validation become the driving force. Furthermore, besides the loose-coupling through interfacing and composition, the formal framework we are envisioning must intrinsically support the above advanced service features, namely persistency, knowledge-intensiveness, distribution and adaptability. Besides that, in complementing the substantial current efforts aimed at backwardly formalising service deployment standards, we instead pursue in this thesis a forward methodology that keeps the domain level completely independent of any specific service-technology solutions, which opens the quest for best deployment alternatives with respect to all available service technology and related proposals. Moreover, Our research method is a typical theoretical research technique in which we built a model, formalise it and then prove properties about it. It is not empirical and not qualitative.

1.2 Research Questions

In light of the above motivated research direction and scope we are pursuing during this thesis, we formulate in the following, in a more precise manner, the main research questions we are tackling. Furthermore, to explicitly motivate for the methodology we have been undertaking to resolve this main thesis's research question, we refine it into several manageable sub-questions.

Why to Discuss Conceptual Modelling:
A conceptual model is a visual method that describes 'concepts' (entities) of the proposed system and relationships between them. Also, it is a well known technique of data modelling which can be independent of the implementation detail. However, modelling are used to analyse, understand and prove desirable properties of the system before it belt.

Thesis Research Questions

How to leverage the development of realistic service-driven business applications, so that it becomes progressive, comprehensible and yet rigorous, and where distribution, persistency, knowledge-intensiveness, validation/certification and adaptability as well as the harmonious complementarity between local and global *perspectives represent the governing driving forces.*

Towards effectively tackling this main driving research question in a disciplined and progressive way, we clearly require to further refining it into more manageable sub-questions. These ramified sub-questions could be summarised as follows:

(1) At the business level, how best can we cope with adaptability and behavioural knowledge, so they can further be aligned with the business goals of the to-be developed service-driven applications? In other words, which *business mechanisms* are the most suitable to promote adaptability and behavioural knowledge, in an event- and interaction-driven manner as service-orientation strives for?

(2) At the foundation level, among the plethora of available formalisms in software-engineering, which (multi-paradigm) *formalism* scores minimal criteria to be a candidate for formalising and reasoning about the above advanced service requirements? More specifically, towards coping with the above enumerated features/requirements in advanced services, we impose on any (multi-paradigm) formalism to be a candidate at least the following minimal conceptual criteria.

Understandability Through Visualisation: Experience shows that formalisms endowed with graphical descriptions are more accepted by cross-organisation stake-holders (not just designers and programmers). Furthermore, understandability is essential in bridging the gap with the business-level, where first intuitive description of the service-driven application, at hand, is given in terms of global goals and processes.

Concurrent and Distributed Behaviour: As we pointed out distribution is one of the main characteristics of (advanced composite) services. Consequently, the targeted formalism intrinsically supports concurrency and distribution.

Type- and Instance-Level Support: We argue that one of the shortcomings of Web standards, such as WSDL, WSBPEL and WSCI, is their inability to intrinsically cope with the instance-level, where one may directly address and reason about specific services. Moreover, coping with both the type- an instance-level thus represents a critical requirement to tackle service states, and thereby explicitly dealing with persistency and conversation.

Validation and Verification: As we are endeavoring bridging the gap between the business-level and the conceptual level, requirements validation are thus essential. The validation should thus include: requirements missing, misconception, misunderstanding, conflicts and so on. Graphical validation in this respect is highly requested. Besides such validation, the formal verification of crucial properties are also to be taking into account.

Advanced Abstraction Mechanisms: On the one hand, we argue that current monolithic interfacing is not sufficient to offer tailored features to different customers and/or focussed compositions. Object-oriented advanced mechanisms such as classification and aggregation seems to be very beneficial for supporting any promising candidate formalism. On the other hand, the ability of addressing componentisation and explicit inter-component interactions represent prerequisites to cope with service compositions via (structurally and behaviourally) rich interfaces.

Semi-Formal Standards Compliance: Since UML is the de facto standard while developing complex (cross–organisational) information systems, it is highly beneficial that some features of the envisioned formalism could be smoothly derived from specific UML diagrams and/or supporting OCL constraints [81].

(3) Once a candidates general-purpose formalism has been selected, following as much as possible the above criteria. Further research questions directly related to advanced specificities of service orientation should be addressed. These sub-questions should include:

(a) How to leverage such (basic general-purpose candidate) formalism so that it can explicitly cope with *service* interfaces specification and validation?

(b) How to adapt it so that it can explicitly integrate knowledge-intensiveness, and in particular *business rules*.

(c) How to leverage such adopted formalism, so that behaviour-intensive service composition, at the local *orchestration*-level can be formalised and reasoned about?

(d) How to bridge the gap between the informal (UML-driven) level with related business rules and the formalised service composition?

(e) How to extend/adapt the formalism, so that one can achieve a harmonious and complementary interaction between such service-focussed orchestration-driven composition and more global and collaborative *choreography*-driven composition?

(f) Last but not least, how to leverage this resulted service-driven formalism for knowledge service-oriented applications, so that we can dynamically adapt and evolve different requirements, specifically behaviour-intensive.

1.3 Original Contributions

Towards the envisioned approach, we have thus been focussing on these research questions. This aimed approach will consequently be in a form of an integrated stepwise conceptual framework composed of the following:

At the intuitive business-level, we are proposing to capture service requirements through tailored UML-based class-diagrams and event-based business rules. At the foundation-level, we are proposing an innovative variant of adaptive high-level Petri Nets, intensively worked out to adequately tackle the emphasised challenges in complex service-driven applications. Referred to as CSrv-Nets, this innovative adaptive high-level Petri Nets conceptual model enjoy most of the above set out requirements and thereby positively fits all the detailed research questions.

Firstly, as a main software engineering formalism, we are adopting a Petri Nets-based conceptualisation. In light of the above service requirements and detailed research questions, the main reasons for such conceptual decisions include the following. Petri Nets [15, 78, 74] are intrinsically graphical, visual and thus highly understandable. Besides being formally defined, on the basis of different operational semantics (e.g. graph-theory [13, 97, 25], logic [98, 6, 8], etc). Petri Nets are inherently concurrent and distributed. They allow reasoning about the generic model as well as its running instances. They permit validation through graphical animation using intrinsic simulations (e.g. token games). They are endowed with several analysis and verification techniques (e.g. place/transition invariants [9] and siphon/trap [85]). Last but not least, since service compositions are business process-driven, Petri nets represent one of the best formalisms to graphically address business processes, concrete workflows , modelling and validation.

Secondly, as first extension of the place/transition nets do not cope with the structural complexity of services, we propose to benefit from high-level Petri nets, such as algebraic [80] and colored Petri nets [47]. More precisely, we first propose to integrate most object-oriented mechanisms (e.g. classification, inheritance and aggregation) into P/T Nets. Moreover, and because the service paradigm is based on explicit interfaces and transient interconnections, we propose a service driven Petri nets variant that involves an object-oriented concept.

Thirdly, since business rules represent the best driving forces for coping with knowledge-intensiveness, we explicitly and soundly integrate ECA (Event-conditions-actions) business rules [41] in the service driven Petri nets variant. On the basis of such explicit integration we demonstrate how process-oriented composition of such behaviour-intensive service interfaces can be progressively conceptualised using the proposed formalism. More precisely, while composing service we explicitly distinguish between service-focussed composition, the so-called orchestration-based composition, and between the global collaborative service composition, the so-called choreography-based composition. We demonstrate how such two compositions are indeed complementary, when the focus are behavioural issues. Thereby, we are leveraging most of the existing approaches for (pure process-centric) service composition, which considers them as two completely independent means of composing services.

Fourthly, by integrating event-driven business rules into the conceptual service-driven Petri nets model, we achieve adaptation but only at the design-time. With the aim of coping with dynamic adaptability of business rules, we finally propose to smoothly and soundly upgrade that conceptual model to allow a dynamic adaptability of its knowledge, based on business rules. We achieve this by endowing each composite service driven Petri nets with an adaptability level, where transition inscriptions can be updated at run-time (i.e. dynamically shifting up/down business rules as advices).

To recapitulate on the achieved contributions, Figure 1.1 illustrates this progressive approach we are putting forwards in this thesis. As we emphasised, the proposed approach is methodologically perceived as being composed of four phases.

UML/Business-rules Requirements Phase: In this preliminary phase, the informal description of the business-driven application at-hand is semi-formally and diagrammatically expressed in terms of UML tailored class-diagrams. Besides that, all related intra- and inter-organisational business rules governing the behavioural features of different basic and composite services are to be clearly described, following in particular the well-known Event-Condition-Action (ECA) paradigm.

Concurrent Services Nets Specification/validation Phase: As we already emphasised, this phase is decisive as it allows to define in a precise and concise way all functionalities and behaviours of different services and their interaction (i.e. service interfaces, elementary and composite services) and validate them against misconception, misunderstandings, conceptual mistakes, etc. For this crucial phase we are proposing a tailored variant of high-level Petri nets, that reflects all structural and behavioural features of elementary or composite services, such as distribution, persistency (stateful), conversation and complex structuring mechanisms (e.g. classification, aggregation). Moreover, we are enhancing the practicability of this conceptual model we refer to as CSrv-Nets, by hiding as much as possible its tedious mathematical side.

Synergetic Complementarity Orchestration/Choreography Phase: Once any CSrv-Nets service behaviour has been specified and validated independently, we give the designer the ability to compose such validated services. This composition is achieved at the global choreographical level yet with a harmonious complementarity with the involved orchestrated CSrv-Nets services. In contrast to the existing process-centric, our composition is knowledge-intensive, thus driven by business rules at the intra- and inter-service levels. In such manner, we are enhancing one step further the *behavioural* adaptability both for elementary services and complex composite services.

Runtime Adaptive Service Nets Phase: This phase allows endowment of the conceptual model CSrv-Nets with an adaptability-level so that adaptivity of different services behaviour can be achieved at-runtime and in a consistent and incremental manner. At that adaptability-level any business rules can be dynamically updated (i.e. added, removed and/or adjusted) independently of the conceptual level. Then, such adaptable business rules are dynamically superposed on the conceptual-level.

Extended WSBPEL for Implementation Phase: Although the thesis does not tackle this phase, we judge it very beneficial to insist that once the conceptual model is specified, validated and dynamically adapted at need, we can straightforwardly derive the corresponding XML-based WSBPEL-like code for orchestration and choreography, namely WSBPEL and WSCI [44, 14]. The corresponding business rules are to be codified using RuleML-like programs. At different stages during this thesis, we give some necessary hints on how such translations could be easily derived. Nevertheless, we should again emphasise that this thesis is about the conceptual level, and thus we do not devote any chapter for that implementation; though we are aware of its crucial importance.

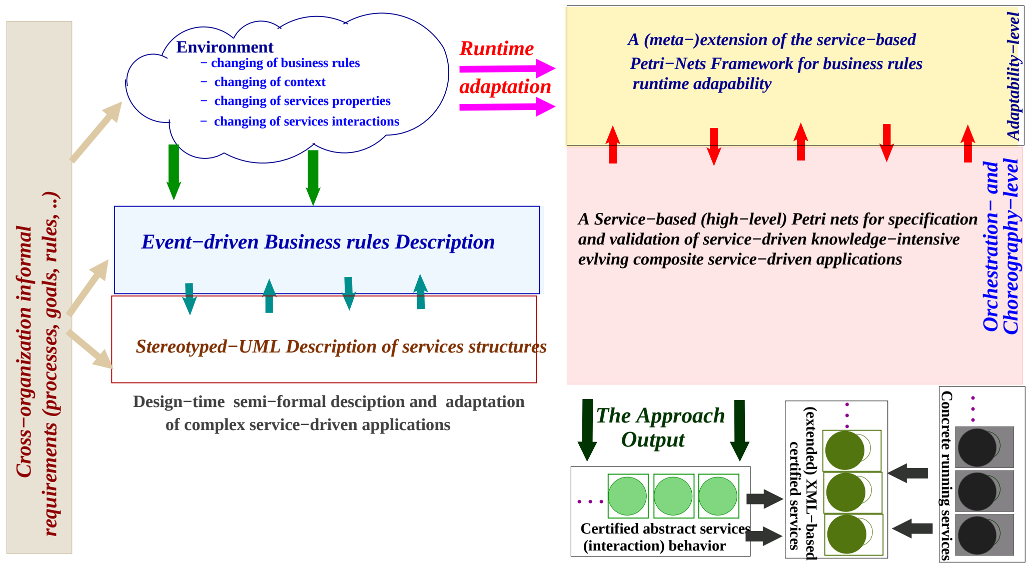

Cross−organization informal requirements (processes, goals, rules, ..)
Environment
− changing of business rules
− changing of context
− changing of services properties
− changing of services interactions
Runtime adaptation
A (meta−)extension of the service−based Petri−Nets Framework for business rules runtime adapability
Adaptability−level
Event−driven Business rules Description
Stereotyped−UML Description of services structures
Design−time semi−formal desciption and adaptation of complex service−driven applications
A Service−based (high−level) Petri nets for specification and validation of service−driven knowledge−intensive evlving composite service−driven applications
Orchestration− and Choreography−level
The Approach Output
Certified abstract services (interaction) behavior
(extended) XML−based certified services
Concrete running services

Figure 1.1: The service-based Petri Nets-based methodology for reliable and adaptable development of complex service applications

Following the proposed conceptual framework and its supported stepwise approach, we demonstrate how we are able to intrinsically and formally address different local and cross-organisational and services business rules. Moreover, to promote competitiveness geared by such business rules, we propose to dynamically adapt such rules depending on the customer profiles, seasons, emerging attractive new regulations, and so on.

1.4 Reader's Guide: Organisation of the Thesis

In accordance with the objectives and targeted contributions of this thesis, we overview in the following the remaining chapters with a summarised content of each one.

Chapter 2: Preliminaries and Service Background

This chapter aims at paving the road for both the topic of this thesis, namely service technology, and at providing the reader with all required basic concepts and background to smoothly follow the subsequent main chapters in a self-contained manner. Precisely, on the one hand, we survey current Web service standards and architectures. On the other hand, as we are pursuing high-level rule-intensive Petri Nets-based conceptualisation.

Chapter 3: High-Level Petri Nets and Web services: Overview and Comparison

Since we are envisioning to formalise and validate complex knowledge-intensive and an adaptive services application using an innovative high-level Petri Nets model, this chapter brings the reader closer to the recently developed approach for formalising Web services using high-level Petri nets. Moreover, after summarising such recent proposals, we put forward a set of criteria for classifying and comparing such proposals.

Chapter 4: Service-based Petri Nets: Foundation, Illustration and Methodology

This chapter of this thesis motivates and puts forward an innovative variant of Service driven Petri Nets for specifying and validating knowledge-intensive service-driven applications (referred to as CSrv-Nets), with particular focus on business rules, distribution and understandability. With respect to scalability and understandability, we demonstrate using the running example, i.e. the travel agency case, how starting from a UML-based informal description, we smoothly shift to CSrv-Nets formalisation of the service-driven application at-hand.

Chapter 5: Extensions of Service-based Petri Nets: Harmonious local/global Compositions

This chapter proposes to leverage the introduced CSrv-Nets to cope with the global choreographical perspective. The objective of this chapter consists thus in forwarding a suitable framework based on a sound extension of CSrv-Nets that allows a *harmonious* complementarity between the local service-focussed orchestration perspective and unified global inter-service perspective while composing knowledge-intensive and adaptive services.

Chapter 6: Extending CCSrv-Nets for Dynamic Adaptability

The purpose of this chapter is to go beyond the design-time adaptability of behavioural features governed by event-driven business rules. This soundly extend the conceptual model by endowing its constituents (e.g. interfaces, local and global composite services.) with adaptability level, where rules can be dynamically manipulated. For the dynamic shifting up/down of such business rules of services, we propose inference operational mechanisms.

Chapter 7: Conclusions and Future work

This last chapter first recapitulates on the achieved contributions of this thesis. Secondly, it discusses different possible extensions of this work both at the conceptual and deployment levels.

Chapter 2

Preliminaries and Service Background

This chapter introduces in a self-contained manner the main topics that will be the focus of during this thesis, specifically Web services, as well as all the concepts on which our envisioned approach will be based, mainly Algebraic specification, Petri nets and Business rules. More precisely, in the first section we will overview the main concepts and mechanisms underlying the service-oriented architecture (SOA) and its enabling Web services. In the second section we recall definitions and concepts around business rules and the state-of-art towards adapting Web services using such rules. In the third section we present some formalisms that have been applied to specify service-oriented applications. In the fourth section we present and illustrate the principles of Petri nets and High-level Petri nets. In the last section we recall some of the main concepts of algebraic specifications.

2.1 SOA and Web services: Overview

Web services are a collection of protocols and standards that are capable of exchanging data and messages between applications and heterogeneous systems, etc. They provide a structural way to handle transactions and standard means to describe what the services do and how to make them available to others. Moreover, it uses HTTP by default as a transport protocol to allow messages to be exchanged over the net. Web services are being used as a solution to many of the distributed computing problems and as the future of the electronic business.

Web services in real life can be used in many different applications including, but not limited to: online stores, hotel and airline -booking package, real-time stock market information and banking. Basically, Web Services are very independent, which means ;operating system independent, hardware independent and programming language independent. On other words, it allows applications written in different languages (such as Java, C++, etc) on different platforms such as Microsoft.NET [68], J2EE [46], etc)to be used and invoked by applications running on a remote computer in a standard-based way regardless of differences between the application server technology running on the remote and local machines.

The basic Web service protocol stacks include, i.e. WSDL the descriptions, represented in XML based description languages [83], UDDI which is the entries for describing web services and SOAP [61] messages for actual Web Service invocation and reply.

As the number of available Web services is steadily increasing, there is a growing interest for reusing basic Web Service in a new, composite way. Therefore, many languages for Web services composition have been proposed recently, including WSFL [82], XLANG [93], WSCI [38], and WSBPEL [87].

What are Web services?

As defined by the World Wide Web Consortium [20], a Web service is a software system identified by a URI which designed to support interoperable machine-to-machine interaction over a network. It has an interface that is capable of being described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standard.

In another definition [27], a web service describes a collection of electronic operations that are network accessible through standardised XML messaging. It provides a set of functionalities to business and individual and enables universal to accesses to these functionalities. Web services has moved from tightly coupled system to loosely coupled system using existing web protocols,such as HTTP, SMTP, etc. The development of web services follow similar philosophy to traditional software system. However, web services need to be self contained component that are able to communicate with each other via the web service stack. see Figure 2.1.

The main advantages of using Web services is to allow and perform electronics operations without need for human intervention (software-to-software interaction) and that provides opportunities including but not limited to: reducing the number of employees and the time of integration.

It uses XML document because it has many advantages such as improved data typing and structure, greater flexibility and extensibility. Moreover, XML documents are easy to understand because it was designed as text so that, anyone can always read and configure out the content since it is not in binary data format. To put it in a simple way, web service is a software service that exposed on the web through SOAP, described using WSDL and registered in UDDI.

The Underlying Technologies for SOC

In this section of the thesis, we will discuss Extensible Markup Language (XML) since SOA is an XML-based document. Furthermore, Web services protocol stack will be explained and finally, languages that are used for describing and compositing of Web services will be outlined.

Extensible Markup Language (XML). XML [39] is an extensible mark up language for documents containing structured information. It provides a way to define, store, transmit and a standard way to tag information and the format for data exchanged via web service. It is important to realise that XML is not really a ”language” at all, but a standard for creating languages that meet the XML criteria. In other words, XML describes a syntax that users use to create their own languages. XML schema language provides a way to describe, validate the structure and the contents of XML documents. An XML-document is a structured document containing a top element which is enclosed by a start tag and end tag. Each elements can contain child elements as well as an attributes. XML is a framework for defining markup languages. However, XML allows authors of documents to define their own tags and own document structure for the specific purpose.

Web Service Protocol Stack. Web services stack is a collection of standardised protocols and application programming interfaces (APIs) that let individuals and applications define,locate, implement and utilise Web services [5]. Web services applications are built on an architecture or software system design which can be illustrated as a 'stack'of processing layers. The software components in theses layers are loosely coupled components that interact with one another via standard protocols which have standard interfaces.

The web service stack mainly comprises four areas: communication protocols, service description and service discovery as shown in Figure 2.1

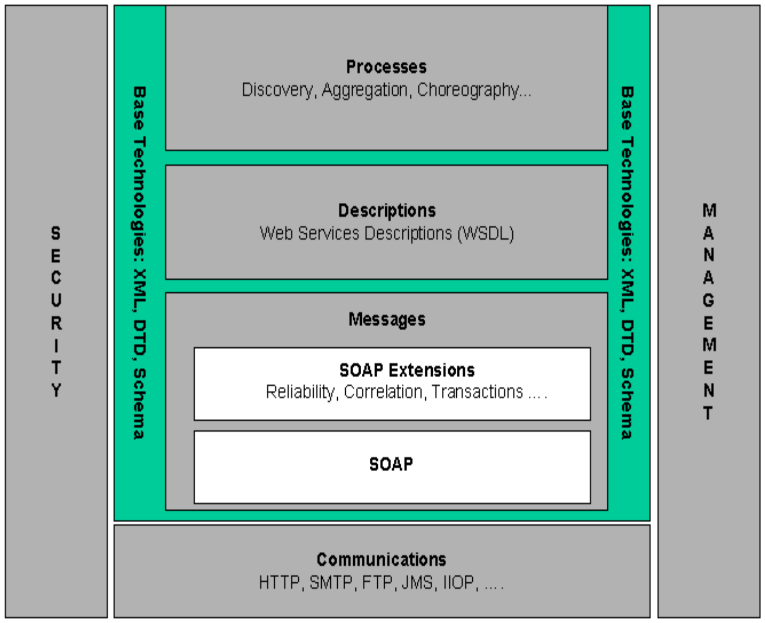

SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
MANAGEMENT
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Figure 2.1: Web services stack

(Service) Communication Protocol

Communication protocol is the foundation layer of the Web services stack which is responsible for transporting messages between network applications.

Web services has to be available over a network to be invoked by a service requester. The network is often based on an HTTP protocol,however this does means that HTTP is the only protocol that can be used.In fact there are other transport protocols that may be used such as SMTP, FTP and HTTPR, etc.

(XML) Messaging Protocol: Simple Object Access Protocol:

(XML) Messaging protocol is responsible for encoding messages in a common XML format so that they can be understood at either end of a network connection. SOAP is an XML-based message protocol, which is specified in a W3C specification. Moreover, the likely technology to provide communication framework for transport XML-based messages anywhere across the net is the SOAP. which facilitate the communication between Web services and their clients. SOAP is the chosen XML messaging protocol for many reasons:

- It is simple; it can be used in combination with or re-enveloped by a variety of network protocols such as HTTP, FTP and SMTP. It is basically an HTTP POST with an XML envelope as payload.
- It is the standardised enveloping mechanism for communicating document-centric messages and remote procedure call (RPC).

SOAP is an XML protocol that facilitate Publish, find, bind, and invoke operations as described previously and its structure consists of the following parts:

- Structure of a SOAP Message: which consists of the following elements:
  - A required envelope element that identifies the XML document as a SOAP message. it contains an optional header and mandatory body.

  - An optional header element that contains meta-information about the message, such as routing, security ,transaction management etc.
  - A required body element that contains the actual payload of the message. (e.g. call and response information) encoded as XML.
  - An optional fault element that provides information about errors that occurred while processing the message

- SOAP Encoding Mechanism: Is another important area of SOAP that is dealing with a set of rules and mechanisms for encoding data in SOAP messages. The SOAP specification's encoding/serialisation portion defines how objects are to be encoded or serialised into a common XML syntax when transmit over SOAP. Having an encoding standard for SOAP messages means that objects can be encoded in SOAP messages in a standard way, and then on the receipt side the message will be decoded. The SOAP library found on the client and the server performs the encoding/decoding. SOAP's encoding/serialisation features are mainly used in conjunction with the RPC mechanism, as explained next.

- Remote Procedure Call on SOAP: RPC stands for remote procedure calls [4] which is the most common messaging pattern in SOAP. RPC is used for making a request message (procedure) or function calls to a server node, and receiving the responses back,both over the network. In other words, SOAP RPC defines the ability to use the SOAP protocol to execute a specific procedures on the server side of a SOAP message. The RPC mechanism builds on the encoding portion by allowing encoded objects to pass as arguments to a remote procedure. The only thing that has to be known before invoking a remote procedure, is the name of the procedure that intend to call and the arguments to that method.

(Service) Description Protocol (WSDL)

(Service) Description protocol used for describing the public interface to a specific web service . Web Service Description Language (WSDL) is the layer above XML-based messaging protocol which is a specification that describes available Web services to requesters. These descriptions written in XML form and describe the public interface and implementation of Web services.Business can use WSDL to advertise and then expose its services by publishing them in the registry. The reasons why WSDL details have been divided up into two parts (interface and implementation) is to allow each part to be define separately and independently and then reused by other parts.

A WSDL document defines services as collections of network endpoints ports. In WSDL, the service definition is divided up into two parts: *the service interface definition* and *the concrete network definition* for data binding. This enables the reuse of abstract definitions: messages, which are the data typed elements, and port types which are abstract collections of operations.

The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

Types: a container for data type definitions using some type system (such as XSD).

Message: an abstract, typed definition of the data being communicated.

Operation: an abstract description of an action supported by the service.

Port Type: an abstract set of operations supported by one or more endpoints.

Binding: a concrete protocol and data format specification for a particular port type.

Port: a single endpoint defined as a combination of a binding and a network address.

Service : a collection of related endpoints.

In addition, WSDL defines a common binding mechanism with SOAP, HTTP and MIME This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. It allows the reuse of abstract definitions. Finally, WSDL provides the foundation on which Web Service Composition Languages build up on.

(Service) Discovery Protocol

UDDI is a XML-based global, public or private online directory which enables business or individuals to list businesses that they provide as a service provider and to be discovered by other services around the global.

UDDI is extensible data model that designed to work with any service description language such as WSDL. Services in the registry have to be well classified according to their functionality in order to allow easily requesters to find services that satisfy their needs.The UDDI specification consists of an XML schema [96] for SOAP messages and a description of the UDDI APIs specification.The UDDI XML schema defines four types of information in order to use a partners‘s Web service, these are: *business entities, business services,* *binding templates and tModels*. Business entities describes information about businesses including their name, description, services offered and contact information. Business services provides more details on each service being offered. Each service can have multiple binding templates, each describing a technical entry point for the service(e.g. HTTP, SMTP, etc). Finally, tModel describes the particular protocol or standards a service uses.

A registry can run by a various vendors such as IBM and Microsoft. Registration allow business's publisher to obtain an authentication token and to log onto an operator‘s site to post its information via SOAP.By doing so it will allow businesses to find each other.

Publication of a services is the action that performed by a service providers to allow the description of the services available to a potential service requesters. Also, publication refers to advertising the WSDL in a UDDI registry. Like publishing web service descriptions, discovery of any service is the action that gives the service requester access to WSDL for a service.requesters are able to query the registry to view and retrieve information regarding to a specific service that fulfill their needs and depending on some criteria they decide whether to invoke the service or not.

2.1.1 Services-Oriented Architecture

Traditionally, there are a number of architectural styles that have proposed to build and enhance the performance of distributed systems, such as DCOM [67] or CORBA [71]. Nowadays, the current trend in the application development is shifted towards loosely coupled and browser-based applications, due to the fact that, such clients eliminate the high costs of deploying an applications to the desktop. Desktop applications are costly to deploy partly due to the issues of instation and configuring the application and partly due to the issues of communicating between the client and the server. Therefore, HTTP is one of the communication transport protocols that is used as a solution to many of the distributed computing problems and as the future of the electronic business. HTTP is a good choice because any machine that can use a web browser is by definition running HTTP. The latest evolution in this new category of systems is a new paradigm, called Service Oriented Architecture(SOA). SOA is an architectural style which has been proposed recently that has promising functions to internet based business applications. This represents software functionality as services which can be described ,located and invoked over the network.

Web service architecture describes three basic roles: service provider, service requester and service broker; and three basic activities: publish, discover, bind and invoke [36]

Service Provider: From a business perspective, this is the owner of the service and from an architecture perspective; this defines the service details and then publishes it to one ore more repositories (service registry) based on standard called UDDI for potential users to locate.

Service Requester: From a business perspective; this is a business that requires a certain functions that satisfy it needs. From an architecture perspective, this application will discover, bind and initiate an interaction with a specific service. The Service requester could fall down into different categories such as a human user requesting a service via browser-based interface or an application program or it could be another web service.

Service Broker: This provides a searchable repository of service description where service providers publish their services descriptions. service requester find services and request access to those services by creating binding to the service provider.

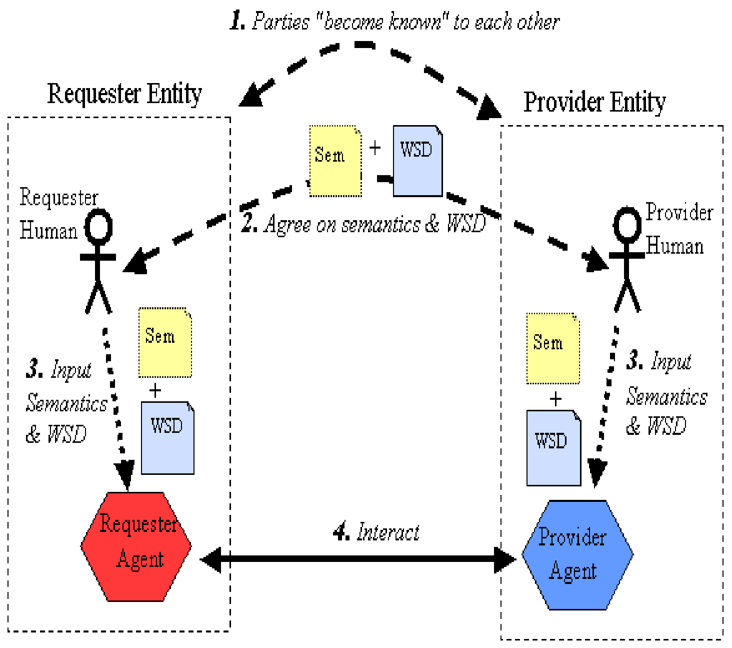


Figure 2.2: Web services Architectur (W3C Architecture Working Group)

Languages for Web services Specification and Composition

Originally, Web services provide the basic technical platform required for application interoperability. They do not, however, provide higher level control, such as which Web services need to be invoked by an application programs or other services, which operations should be executed and in what order. Nor do they provide ways to describe the semantics of interfaces, the workflows, or business processes.

Since the interaction model that supported by WSDl is essentially a stateless model because the language is not aware of states in-between operation, and business collaborations require long running interaction driven by an explicit process model. Therefore, there is a need for a formal ,richer description languages to fill the gap between mere interface definition language (e.g. WSDL that do not support any complex and flow process).

Moreover, Web services composition languages have to support a set of minimal requirements before the full potential of Web Service Composition reliased. The five basic control patterns described by Aalst in [3, 88] are:

Sequence : Which defines the activities execution order.

Parallel Split: Which is a point in the work flow process where a single activity splits into a multiple of activities which can be executed in parallel.

Synchronisation: Where multiple parallel execution paths joins together by waiting for all of them to finish before flow continue.

Exclusive Choice: A point in the work flow process where, based on a decision or workflow control data, one of several activities is chosen.

Simple Merge: will only wait for the first out of many execution paths to finish, before flow continue.

Recently, many specification languages have been proposed for the past years,which are aimed to support different aspects of web service interactions. including WSFL, XLANG, WSCI and WSBPEL, etc.This section discusses briefly a number of Web services *Description* and *Compositional* languages along with a brief comparison between them.

Web Service Flow Language (WSFL): This Language was created by IBM. It is an XML-based language which provides the mechanism to deal with complex interactions between one or more services that act both as clients and servers,Thus provides a support for a business process. WSFL represents a business process as graphical-oriented model (flow model), which is a visual representation that make it easy for the end-users to understand and communicate. Graphs defined using a set of activities(tasks. A flow model is an abstract definition of the work flow process. It describes a usage pattern of a collection of a available service, so that the composition of those services provides the functionality needed to achieve a business tasks,that is, the composition describes a business process.

A flow model contains *Activity* elements, which define a sets of an individual business tasks that have to be performed as a part of a business process. *Control Link* specifies the execution order of the individual business tasks to be performed.*Data Link* specifies the flow of data from one activity to another. *Service Providers* are abstractions for the concrete business partners with which the flow model interacts.*Service Provider type* is defined by a WSDL portTypes, which represents a service(s) that a service provider(s) will participate in the flow. Next to the flow model there is a global Model which defines how a given business process is implemented. Such as identity, location and the implementation of the service provider that implement a specific task. once the global model and the flow model for a given business process are defined, the completed business process can be exported as single Web service , that is, by making them available for direct interactions by other business process or service clients.

Web services Business Process Execution Language (WSBPEL). It is a modified version of Business Process Execution Language for Web services (BPEL4WS),which has been recently defined to describe compositions of services based on the business process model. WSBPEL specification builds on IBM‘s WSFL (Web Service Flow Language) and Microsoft‘s XLANG (Web service for Business Process design) which allows a mixture of Block structured and graph structured process model.

WSBPEL supports *sequencing of peer-to-peer message exchange, both asynchronous and synchronous, within a stateful interactions involving two or more parties.*.However, Business processes can be described in two ways:

- Business protocols (Abstract Process), in contrast, use process descriptions that specify the mutually visible message exchange behaviour of each of the parties involved in the protocol, with the hiding of their internal behaviour.

- Executable business processes (invocable Process) that models actual behaviour of a participant in a business interaction. In other words, A business process defines how a process instance coordinate the interactions with its partners.

WSBPEL is used to model the behaviour of both executable and abstract processes. The scope includes:

- Sequencing of process activities, especially Web Service interactions.

- Correlation of messages and process instances.

- Recovery behaviour in case of failures and exceptional conditions.

WSBPEL fits into the core Web service architecture since it depends on the following XML-based specifications, XML Schema, WSDL and XPath. In this sense, a WSBPEL process definition provides and/or uses one or more WSDL services, and provides the description of the behaviour.WSBPEL process definition, defines data variables, partners and process flow construct. A partner link type characterised the conversational relationship between two services by defining the ”roles” played by each services in the conversation and specifying the port types provided by each roles.

It is worth to finally mention that current web content is designed primely for human to read and not for machine to understand. Moreover current Web services standards such as WSDL, UDDI and WSFL are not semantic-Oriented, therefore, there is a need to remedy this disadvantages and to bring more meaningful information embedded into the web content by combining semantic [12] to the Web services. The realisation of the so-called Semantic-Web services is underway with the development of new AI-inspired content markup languages, such as DAML-S ,WSMO [45], etc. These languages have a well-defined semantics and enable the markup and manipulation of complex taxonomic and logical relations between entities on the Web. By doing so will enable automated machine data processing such as discovery, negotiation, interaction and to minimal human intervention.

2.2 Business Rules and Web Services

Although service-orientation strives for flexible composition of complex Web services from simple ones, the current technology and standards simply does not allow such adaptable composition. In WSBPEL, for instance, as one of the most representative language for composing services, the corresponding business process (inter-related business activities, operations, conditions and messages) must be *predefined* in advance at the specification time and remains *static* (i.e. non adaptive) during time execution.

This inability of *automatically*[1] adapting Web services behaviour induces severe problems for this technology to deliver all its promise.

- The lack of adaptation prevents modifying composition behaviour (e.g. activities ordering, operations, conditions, etc) at runtime as customers or other participating partners change or introduce new requirements.

- As Web services functionalities unpredictability and rapidly change to stay competitive, static composition implies very often working on obsolete and out date version of statically selected Web services.

- Static composition makes very difficult dealing with optimal composition, as quality of services and optimal performance implies runtime assessment.

In the rest of this section, first, we report on business rules as one of the main means to address adaptability. We then present some ongoing approaches for adaptable Web services using business rules.

2.2.1 Business Rules Concepts

Business rules [84, 48, 52, 55] have been recognised as the main driving means towards *adaptable* information systems. Business rules can be defined as "projections of organisations' constraints and declarations of (internal/external) policy/conditions that must be satisfied for doing *business*". They plays a crucial

[1]Due to the huge complexity of service applications any *static* adaptation is hard, error prone and too slow to be effective.

role in determining how operational decisions within or between organisations must be made. In particular, business rules specify actions on the occurrence of particular business events, including 'state of being' changes concerning individuals and groups of individuals, infrastructure, informational resources, and business activities. They inform about guidelines and restrictions with respect to states and processes in an organisation. Therefore their collection, expression, structuring and organisation have been acknowledged as central activities within any business/software model.

The Pionner work of M. Loucopoulos [49] distinguishes between 'Intentional' and 'operational' business rules. Intentional rules are expressions of business rules seen from a *business context perspective*. They express laws, external regulations, or principles and good practices specifying the way an organisation conducts business and are usually expressed in the form of *natural* language statements. 'Operational rules' are expressions of business rules approached from a *business process perspective*. They prescribe action on the occurrence of some business events, or describe valid states of an organisation's information entities. Operational rules generally derive from the translation of informal intentional rules to formal rule statements developed in accordance with a convenient rule language and conceptual schema, also making reference to other enterprise knowledge concepts (e.g. actors, activities, activity enablers, information objects and attributes).

Business rule-driven business models enjoin, therefore, very determinant advantages for coping with dynamically evolving complex business processes. First, they are specified independently of processes so they are intrinsically evolvable. Second, they focus on more primary business-driven requirements are thus need to be tackled at early phases of business applications. Last but not least, they respect declarative descriptions rather than specific operational ones, which opens different way of abstractly conceiving and validating them.

2.2.2 Adapting Web Services through Rules

Business rules are thus generally conceived to be evolving. Unfortunately, with respect to service-driven business applications, they remain almost unexplored preventing this paradigm from all these business processes modelling potentialities. Nevertheless, very recently there have been a growing interest to business rules for endowing Web services compositions languages (specifically WSBPEL), in particular, with the required agility to leverage them to the promised *dynamic* and automatic composition. In the following we sketch these investigations while pointing out some of their strengths to benefit from and some of their shortcomings to be overcome.

- Papazoglou et al. [65] were the first to yield the potential of business rules in service-orientation to endow BPEL with dynamism. This approach proposes a complete business rules-driven life cycle for dynamically composing Web services. Business rules are classified based on the requirements of service composition, instead of general usual classification appeared in [48]. In this approach, starting from a very general specification, the composition is scheduled, constructed and finally executed with the assistance of business rules judiciously classified in a repository. By raising the level of abstraction compositions developed in our approach are flexible and agile in the face of change. Besides basic elements such as events, conditions, and messages, this classification includes rules dealing with the activity flows, the data required for their composition and the constraints to be respected. The direct construction and subsequent, execution of the composition from the business rules is performed in terms of XML-like descriptions.

- In [19], the authors present a more pragmatic hybrid approach for realising the integration of business rules (modeled as aspects) with a BPEL orchestration engine by using aspect-oriented programming techniques [26]. Their approach, called AO4BPEL, is an extension to BPEL by aspect-oriented concepts which allow to model business rules as aspects and weaving them into the BPEL code by using an aspect-aware orchestration engine. The idea is thus to split BPEL into business-flow and business rules. This approach allows business rules to be specified/evolved independently of the (reduced) BPEL descriptions. The integration of both business rules and business-flow is achieved using aspect-oriented programming . This vision fits well with our ideas and can be exploited for further concretisation or implementation of our approach.

- As stated in [21], context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Despite these interesting business rules-driven proposals, we argue that the exploitation of business rules potentials for achieving new degrees of dynamism and abstraction in Web Services composition remains largely unexplored. More particularly for Web services adaptivity, we argue that a more rigorous setting that integrates business rules in the service (interface) behaviour, as the one we are striving to put forward, seems still to be overwhelmingly needed.

2.3 Formalisms for Modeling Service Behaviour

Service Composition refers to the process of creating complex services from a set of simpler, easily executable and lightweight service components. It is similar to component or object based software design and implementation, however differs in its approach of composing and binding services. A typical example of an integrated service is an e-learning system that uses content management system for delivering the courses and tutorials, discussion forums, message boards and voice interactive applications as components. The component services maybe outsourced to various universities and these universities can in turn outsource part of all the activity invoked by delivery of services to different tutors. Web services aim at replacing large monolithic applications with modules of smaller services that can be coupled together to meet the functionality and interpretations desired by the user.

Some of the major artifacts of software engineering that need to be considered while composing services are:

- Problem Domain: This encompasses knowledge of the environment. The domain knowledge must be consistent or satisfiable

- User Requirements: This states what the stakeholders need from the system in terms of desired effects on the environment. A description of the system in conjunction with the environment must satisfy the expected requirements.

- System Description: This includes description of the behaviour of the system. For every possible behaviour of the environment, there must be a behaviour of the system that is consistent with the system description.

The main reasons to use formal modeling techniques for modeling Web Services behaviour, can be pointed as follows [73]:

- They provide us with a fully understanding of the behaviour of any service of interest.
- Analysing requirements by introducing tools to analyse protocol models and find out facts and errors (validation, verification) both at the specification level and the implementation level.
- Justifying design decision and result by construction in a correct and flexible Web services implementation.

Formal techniques to Web service composition can be broadly classified as:

- AI Planning
- Process Algebra
- Logic Based
- Petri Nets
- Workflow Techniques

In the following sections, we introduce of some techniques that are currently proposed for formally modelling software-intensive systems behaviour in general and Web services more particularly.

2.3.1 Interval Temporal Logic (ITL)

Interval Temporal Logic(ITL) [16] is a flexible notation for both propositional and first-order reasoning about periods of time found in descriptions of hardware and software systems. Unlike most temporal logics, ITL can have many features such as the capability of handling both sequential and parallel composition, offer extensible specification and proof techniques for reasoning about properties involving safety, liveness and projected time. Timing constraints are expressible and furthermore most imperative programming constructs can be viewed as formulas in a slightly modified version of ITL. Tempura provides an executable framework for developing and experimenting with suitable ITL specifications. Also, ITL and its mature executable subset Tempura have been extensively used to specify the properties of real-time systems where the primitive circuits can directly be represented by a set of simple temporal formula. In addition, various researchers have applied Tempura to hardware simulation and other areas where timing is important.

In [89] a very promising formalisation of Web services and semantic Web have proposed using an appropriate variant of ITL. Without going into detail about this interesting approach, first (basic) service behaviour is capturing as an ITL specification. Then, using the strong compositionality of ITL complex service are specified and verified using existing tool of this ITL formalism.

2.3.2 Finite State Process Models (FSP)

As Introduced in [28], The FSP notation is designed to be easily machine readable, and thus provides a preferred language to specify abstract workflows. FSP is a textual notation (technically a process calculus) for concisely describing and reasoning about concurrent programs. The constructed FSP can be used to model the exact transition of workflow processes through a modeling tool such as the Labeled Transition System Analyser(LTSA), which provides compilation of an FSP into a state machine and provide a resulting LTS. LTSA is a tool which provides a means to construct and analyse complex models of finite state process specifications. This tool provides the opportunity to model workflows prior to implementation and deployment testing, and with the message sequence chart extensions to easily model workflow scenarios.

In this [29] these capabilities of FSP have been extensively exploited, where service composition using BPEL were first specified and validated using a suitable variant of FSP. An advanced FSP-based tool have been implemented for supporting an automated checking of different properties of the composed services using BPEL.

2.4 Petri-Nets (PN)

In 1962, Petri Nets was first created by a mathematician named Carl Adam Petri in his PhD thesis ”Kommunikation mit automaten” [75]. Here are many reasons that make Petri nets one of the leading framework for describing and analysing behavioural aspects in different kinds of systems.

- They have been used to model and analyse complex applications in a wider variety of domains such as distributed software systems for modeling (business) process, telecommunication for designing and analysing protocols,banking system, etc.

- They sharply distinguish between states and activities (the latter defined as state changes), through the distinction between places (local states) and transitions (local activities).

- Depending of the chosen interpretation different semantics can be assignment to the behaviour of a Petri net ranging from sequential, interleaving to true concurrent ones.
- While being formal, Petri nets also come with graphical representation (i.e. states can be modeled as circles, operations as boxes, and flow relations as an arcs).

Place/Transitions Petri Nets:

Place/Transitions nets is a Petri net comprising a net graph with positive number associated with arcs and an initial marking function which associate a natural number of simple tokens'black dot' with places. The definition of **(Place/transition-Petri nets)** can be represented by a tuple $N = (P, T, F, M, W)$ where :

(i) P and T are nonempty, finite, disjoints sets (the *places* and *transitions* of N, respectively),

(ii) $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs (flow relation),

(iii) $W : F \rightarrow N/0$, attached a weight to each arc of the net,

(iv) $M : S \rightarrow N$, is the initial marking.

Places, transitions, and arcs will graphically be modeled by circles, boxes and arrows, respectively. We also mention that for some practical cases, capacities may be attached places. Each capacity represent a natural number as a maximum number of tokens to be hold in such place.

▶ **Example 2.4.1 (the dining philosophers)** Figure 2.3 shows the well-known example of the five dining philosophers. In the left hand side of the net each philosopher Pi (with $i \in \{1, .., 5\}$) may be in one of the two states, either eating or thinking, corresponding respectively to (presence of a token in) the places Pi_E and Pi_T. Each fork is modeled by a corresponding place, where the presence of a token indicates the availability of the fork. When philosopher's state changes from thinking to eating (resp. eating to thinking), the two forks on its left and right become no more available (resp. available again). Initially, all philosophers are thinking and thus all forks are available.

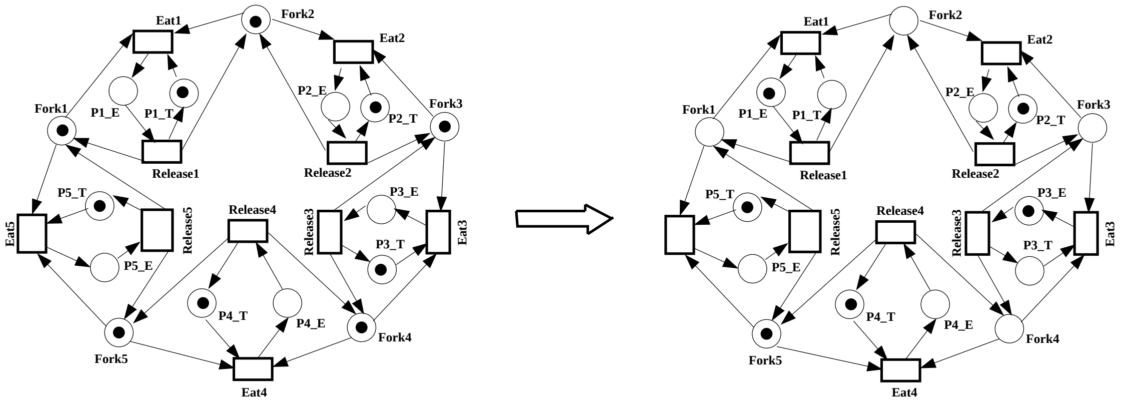


Figure 2.3: The dining philosopher problem as a P/T-net.

Definition 2.4.2 (Transition enabling and next marking)

(i) Given a transition t, its input places are represented by ${}^{\bullet}t$ while its output places are represented by $t^{\bullet}$. They are formally defined by:

${}^{\bullet}t = \{p \mid (p, t) \in F\}$

$t^{\bullet} = \{p \mid (t, p) \in F\}$

(ii) A transition t is M-enabled (i.e. it can be fired under the marking M) iff $\forall s \in {}^{\bullet}t : M(p) \geq W(p, t)$.

(iii) An M-enabled transition $t \in T$ may yield after its firing a follower or next marking M' of M which is such that for each $p \in P$,

$$M'(p) = \left\{ \begin{array}{l} M(p) - W(p,t) \quad \texttt{iff} \quad p \in {}^{\bullet}t/t^{\bullet} \\ M(p) + W(t,p) \quad \texttt{iff} \quad p \in t^{\bullet}/\,{}^{\bullet}t \\ M(p) - W(p,t) + W(t,p) \quad \texttt{iff} \quad p \in {}^{\bullet}t \cap t^{\bullet} \\ M(p) \quad \texttt{otherwise} \end{array} \right\}$$

► **Example 2.4.3** By applying these firing rules to the initial marking of the dining philosopher net in the left-hand side of Figure 2.3, we may for instance result in the marking depicted in the right hand-side. This net is resulting from firing the transition `Eat1` and `Eat3`, that is, the first and the third philosophers enter the eating state while their left and right forks (i.e. f_1, f_2, f_3, f_4) become no more available.

Finally, it is important to point out that in the next chapter, we will go in detail about the application of the (high-level) Petri nets for the formal specification and validation of Web services.

2.5 Algebraic Specification: An Overview

Abstract data types are ubiquitous in different programming and specification paradigms. They allow describing a class of data domains. It is therefore desirable to find away of characterising their semantics. The most widely accepted methods of describing abstract data types use *many-(order-)sorted algebras*. This section introduces some standard definitions and results of many-order-sorted algebras and algebraic specifications. Most of these definitions are borrowed from [24, 30].

Definition 2.5.1 (Order-Sorted Signature) An *order-sorted signature* $Sig = (S, \leq, F)$ consists of a set S of *sort names*, a partial order $\leq$ on S, and a set F of declarations of *function symbols* with arity in $S^* \times S$. The elements of S^* are often denoted by $\vec{s}$. If there are several declarations $(f : s_1 \times ... \times s_n \rightarrow s_0) \in F$ for the same symbol f and different arities $s_1 \times ... \times s_n \rightarrow s_0$ then f is called *overloaded* in *Sig*. If $\leq$ is the flat ordering (i.e. for no $s_1, s_2 \in S, s_1 \neq s_2$, we have $s_1 \leq s_2$), then Sig is called *many-sorted*.

▶ **Example 2.5.2** Throughout this thesis, we follow the OBJ syntax language [31]. The following algebraic signature, for instance, defines the syntax of stacks.

```
obj Stack is .
  sort Item .
  subsort Empty-Stack   NonEmpty-Stack < Stack .
  op :  i0 :  → Item .
  op ∅ :  → Empty-Stack .
  op next :  Item → Item .
  op push :  Item  Stack → NonEmpty-Stack .
  op pop :  NonEmpty-Stack → Stack .
  op top :  NonEmpty-Stack → Item .
endo .
```

Definition 2.5.3 (Order-Sorted Algebras) Let $(S, \leq, F)$ be an order-sorted signature. Then a universe order-sorted algebra consists of a carrier $\mathcal{A}$ together with an indexed set $\{\mathcal{A}_s \mid s \in S\}$ of subsets of $\mathcal{A}$ such that $\mathcal{A} = \bigcup_{s \in S} \mathcal{A}_s$, with an interpretation of each operator $(f : s_1 \times .. \times s_n, s) \in F$ as a partial function $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times .. \times \mathcal{A}_{s_n} \rightarrow \mathcal{A}_s$ such that:

- $\mathcal{A}_s \subseteq \mathcal{A}_{s'}$ for $s \leq s'$, and

- if $(f : \vec{s_1} \rightarrow s_0) \in F$ and $(f : \vec{s'_1} \rightarrow s'_0) \in F$ and $\vec{s_1} \leq \vec{s'_1}$ then $f^{\mathcal{A}} : \vec{\mathcal{A}_{s'_1}} \rightarrow A_{s'_0}$ restricted to $\vec{\mathcal{A}_{s_1}}$ equals to $f^{\mathcal{A}} : \vec{\mathcal{A}_{s_1}} \rightarrow \mathcal{A}_{s_0}$.
- the class of all Sig-algebras is denoted by $Alg(Sig)$.

Definition 2.5.4 (Variables) We assume an infinite set X of variables. A metavariable over X is denoted by x. All variables are typed when used. The type of a variable is made known by a *declaration* of the form: $x : s$ the sort of x is denoted $sort(x)$.

Definition 2.5.5 (Terms and Term-Algebra) Let $Sig = (S, \leq, F)$ be a signature.

(1) A *term* of sort $s \in S$ is either a variable x with $sort(x) \leq s$ or it has the form $f(t_1, ..., t_n)$, where $(f : s_1 \times .. \times s_n \rightarrow s_0) \in F$ and t_i is as term of sort s_i and $s_0 \leq s$. A term over Sig is a term of sort $s \in S$.

(2) A term is called *closed* (or *ground*) if it contains no variables, otherwise it is *open* (or simply a term).

(3) The set of all open terms over Sig with variables from a variable set X is denoted $T_{Sig}(X)$. The set of open terms over Sig of sort s with variables from X is denoted $T^s_{Sig}(X)$. The set $T_{Sig}(\emptyset)$ of all closed terms is denoted T_{Sig}, and the closed terms of sort s are T^s_{Sig}.

(4) T_{Sig} is a Sig-algebra that takes: $(T_{Sig})_s := T^s_{Sig}$, $f^{T_{Sig}}(t_1, .., t_n) := f(t_1, .., t_n)$. T_{Sig} is called the *term algebra* of Sig.

Term algebras seem trivial but are fundamental because they reduce the universe to things that are nameable by the closed terms of the signature.

Definition 2.5.6 (Assignment and Substitution) Let $\mathcal{A}$ be a Sig-algebra and X be a set of variables.

(1) An *assignment* $ass : X \rightarrow \mathcal{A}$ is a function with $ass(x) \in \mathcal{A}_{sort(x)}$.

(2) Let t be a *Sig*-term containing only variables from X. The *denotation* $ass(t)^{\mathcal{A}}$ is defined by induction on the structure of t:

  (a) $ass(x)^{\mathcal{A}} = ass(x)$,

  (b) $ass(f(t_1, ..., t_n))^{\mathcal{A}} = f^{\mathcal{A}}(ass(t_1)^{\mathcal{A}}, ..., ass(t_n)^{\mathcal{A}})$. If t is a ground term we write $t^{\mathcal{A}}$ instead of $ass(t)^{\mathcal{A}}$.

  (c) A *substitution* is an assignment $\theta : X \rightarrow T_{Sig}(Y)$ for a set Y of variables.

Definition 2.5.7 (Equations) Let $Sig = (S, \leq, F)$ be a signature. An *equation* over *Sig* has the form (D, C, e), where:

$D = \{x_1 : s_1, ..., x_n : s_n\}$ for $s_i \in S$ is a set of typed variables declarations,

e is a pair of terms t_l and t_r, written : $t_l = t_r$, and

C is a set of pairs of terms of the same form as e called *conditions*.

All variables occurring in C and e must be declared in D. If C is empty, the equation is called *unconditional*, otherwise it is called *conditional*. e and each of the element of C are called *open equations*. If the set of variables declared in D is X, then we also write $D(X)$ instead of D.

2.5.1 High-level Petri Nets (HLPN): An Overview

Basically, one of the main drawback of using petri-nets is the explosion of the size of their graphical elements when describing complex systems. Therefore, High-level Petri nets were developed to overcome this problem by introducing higher-level concepts, such as the use of complex structured data as a tokens with information attached to them. High level is the standard term that used in the Petri nets forum. Also, High Level Petri Nets is formal because they are mathematically defined, with the possibilities of different semantics (e.g. interleaving, steps, concurrency) depending on the specificities of modelled applications.

Algebraic Petri Nets

High-level Petri nets [43] and algebraic Petri nets [79] in particular have been mainly introduced to significantly reduce the size explosion of Place/Transition nets when dealing with real complex systems. Algebraic Petri nets thus support the construction of consice, but nevertheless comprehensible and transparent models of real-world systems. The main ideas consist in *gathering* different places referring to a same kind (or sort) of entities, where instead of black dots tokens we result rather in algebraically structured ground terms. Given a Place/ Transition net this operation returns to factor out all common similar subnets also called a folding operation. With such structured tokens also arc inscriptions have to be adapted in consequence; they are in general multiset of terms with a same sort corresponding to their input/output places. Transitions' firing involves different notions of term substitutions.

The purpose of this subsection is to recall the main concepts of algebraic Petri nets as introduced in [79]. First, the notion of multiset of terms which represents the key element in algebraic Petri nets is introduced. Then, we recall the formal definitions of algebraic Petri nets.

Given a specification $SPEC = (S, OP, E)^2$ [2], we denote the specification **m**_$SPEC$ by $(\widehat{S}, \widehat{OP}, \widehat{E})$, respectively. Following the OBJ notation, **m**_SPEC can be described as follows:

```
obj m_SPEC is
  extending SPEC .
  sort m_s .
  op ϑ_s :  → m_s .
  op MAKE_s :  s → m_s.
  op _+_s _ :  m_s  m_s → m_s.
  op -_s _ :  m_s → m_s.
  var t_1, t_2, t_3 :  m_s
  eq t_1 +_s ϑ_s = t_1
  eq t_1 +_s t_2 = t_2 +_s t_1      /* the commutativity of +_s */
  eq (t_1 +_s (t_2 +_s t_3)) = ((t_1 +_s t_2) +_s t_3)      /* the associativity of +_s */
  eq t_1 +_s (-_s t_2)) = ϑ_s
endo.
```

For sake of simplicity, in the following with multiset terms we will drop the sort indices s of operations symbols, and write ϑ instead of ϑ_s. As an example, with constant symbols a and b of some sort s, $a-b$ for instance will stand for the multiset terms $MAKE_s(a) +_s (-MAKE_s(b))$. Nonnegative multisets can be specified using (besides the operation symbols of the underlying specification) only the operation symbols ϑ_s, $MAKE_s$ and $+_s$. This motivates the following concepts.

Definition 2.5.8 (Algebraic Petri nets) Let $N = (P, T, F)$ be a net, let $SPEC = (S, OP, E)$ be an algebraic specification, and let X be a family of Sig-variables—with $Sig = (S, OP)$.

[2] We are using (S, OP, E) instead of (S, F, E) as previous due to the use of F as arc relation.

(i) A mapping $s : P \rightarrow S$ is called a *sort assignment* of N. Assuming s, for places $p \in P$ let $\widetilde{p}$ denote the multiset sort $m_{s(p)}$.

(ii) A mapping $M_0 : P \rightarrow T_{OP^+}$ with $M_0(p) \in T_{OP^+,\widetilde{p}}$ for each $p \in P$ is called a s-respecting initial marking of N.

▶ **Example 2.5.9** (The dining philosophers as an algebraic net) A simple look at the dining philosophers modelling in Figure 2.3 using P/T-nets shows that this net is composed of five similar subnets that could not be reduced in a one subset due to the indistinguishability of the tokens as black dots. The corresponding algebraic net of this problem as shown in Figure 2.4 achieves such a folding, where all (available) forks are gathered into a single place while philosophers may be either in a 'thinking' place or in a 'eating' place.

To result in a such compact and very comprehensive net depicted in the left-hand side of Figure 2.4 , an associated algebraic specification has to describe the existence of five philosophers denoted by $p_i, i = 1..5$ and five forks denoted by $f_i, i = 1..5$ with *phils* and $forks$ as sorts respectively. It defines also two unary operators Lf and *Rt* representing respectively the left- and the right-hand side forks of a given philosopher; this correspondence is made explicit using two equations. Finally, we note that the sort assignment s to each place is given by s(P_eating)= s(P_thinking) = *phils*, and s(Forks) = *forks*. M_0 and λ are directly depicted in Figure 2.4.

obj `Phil` **is**

  sort `phils  forks .`

  op f_1, f_2, f_3, f_4, f_5 `:` $\rightarrow$ `forks .`

  op p_1, p_2, p_3, p_4, p_5 `:` $\rightarrow$ `phils.`

  op `Lf :  phils` $\rightarrow$ `forks.`

  op `Rt :  phils` $\rightarrow$ `forks.`

```
  var $p_i, x$ : phils, $f_i$ : forks
  eq $Rt(p_i) = f_i$, for $i \in \{1,..,5\}$
  eq $Lf(p_i) = f_{i-1}$, for $i \in \{1,..,5\}$ with $f_0 = f_5$
endo.
```

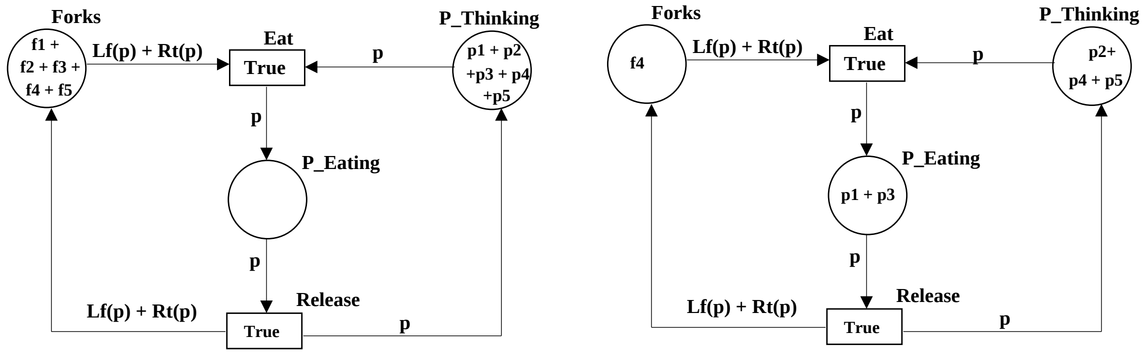


Figure 2.4: The dining philosopher problem as an algebraic Petri net

▶ **Example 2.5.10** In the right-hand side of Figure 2.4 we depicted a next state of the philosophers, where the philosophers p_1 and p_3 enter the eating state; which implies their left and hand-side forks are no more available. To result in this marking, the transition `Eat` has to be fired twice. This first (resp. the second) firing is achieved by substituting the variable p inscribing the input arc relating the place `P_Thinking` to the transition `Eat` by the closed constant term p_1 (resp. p_3). By doing so, the input arc relating the place `Forks` to this transition, namely $Lf(p) + Rt(r)$ is systematically substituted to $Lf(p_1) + Rt(p_1)$ (resp. $Lf(p_1) + Rt(p_1)$), which using the equation in the specification it corresponds to the forks $f_5 + f_1$ (resp. $f_2 + f_3$).

Finally, it is worth to mention that there are many different variants of Petri nets extensions, including Time [10], Cooperative object [86], Object Petri Nets [53, 54], etc.

2.6 Summary

This chapter aims at paving the road for both the topic of this thesis, namely service technology, and at providing the reader with all required basic concepts and background to smoothly follow the subsequent main chapters in a self-contained manner. More precisely, current Web service standards and architectures have been surveyed followed by Algebraic specification, Petri nets and Business rules.

Chapter 3

High-Level Petri Nets and Web services: Overview and Comparison

The purpose of this chapter is twofold. Firstly, an overview of most existing conceptual modelling techniques for Web services based on High-Level Petri nets to be presented in detail. This review is enhanced with the proposition of some exhaustive criteria for assessing and comparing these frameworks with each other, and also to exhibit their strengths and shortcomings (we aim to overcome with our research proposal). Secondly, we present most of the current research investigations for coping with adaptive Web services, and demonstrate how High-Level Petri nets could be a promising alternative for addressing the challenges in modelling adaptivity in Web services (besides distribution, reactivity and conversation, etc.)

3.1 Web Services and (High-Level) Petri Nets

There are currently several ongoing proposals dealing with this crucial area of research and practice, namely the rigorous specification/validation/verification of adaptive composite Web services. Therefore, any attempt toward an exhaustive comparison of such proposals in this area seems to be premature. Moreover, due to the usual divergence in their objectives, formal setting and to the main application domains they are devoted to, it remains very hard to find a common basis for comparing them.

Nevertheless, it is more or less possible to assess existing formal frameworks of Web services with respect to some adequately selected criteria. However, due to the huge number of ways in bringing formalisation to the service-orientation paradigm, we restrict ourselves only to those which are very close to our aimed proposal. More precisely, as the approach is striving to stem from the integration of the component-based paradigm with high-level Petri nets, we will carry out our comparative study with respect to this direction, namely (high-level) Petri-net-based conceptual models for Web services.

In the light of these motivating choices, the rest of this chapter is organised as follows. First, we present an overview of most existing proposals to Web services based on Petri nets, by distinguishing those based on Place/transitions Nets from those based on high-level Petri nets. Secondly, we put forward a set of *"Web service" criteria* that we argue allows assessing the degree of adequacy of any conceptual model as formal setting for service-orientation. Finally, we will apply these criteria to all investigated Petri nets-based models to Web services. This comparison will yield us all limitations and strengths of different existing models, with the ultimate aim to introduce a new form of conceptual model to Web services that overcomes most of their shortcomings and takes profit of their advantages.

3.2 Conceptual Models for Web services Based on Petri Nets

The purpose of this section is to present state-of-the-art existing frameworks based on Petri nets and to address one or more aspects related to Web services modelling and composition. In this study, we distinguish between models based on simple Place/transition Petri nets and high-level Petri nets, which offer more expressiveness and capabilities for intrinsic reasoning more than current Web services technology-based languages.

The first conceptual model using *Place/Transition* Petri nets has been proposed in [34]. It permits specifying the main Web services composition operations. We will also summarise a more specific Place/Transition Nets introduced in [64] for E-service orchestration. Another model based on Place/Transition Petri net was proposed recently in [58, 57] with more emphasise on compatibility of services composition with respect to independent service specification.

Concerning the high-level Petri nets category, the first conceptual model was put forward in [70]. It is based on Predicate Nets and allows modelling semantics Web services languages such as DAML-OIL. A second model based on high-level Petri nets for modelling the flow in Web services was put forward in [69] is based on the so-called Nets in Nets [94]. Finally, the most promising and expressive approach for Web services is the one recently put forward in [99, 100] and is based on Coloured Petri Nets [42].

3.2.1 Web services Models Based on Place/Transitions Petri Nets

In [58, 57], the work is concerned with the application of Web services to distributed, cross-organisational business processes. Each business process is conceived as open Petri nets. Open Petri nets are simple Petri nets with three classes of places: *Input* places allowing messages to get in the process, *output* places to go out from the process, and *internal* places allowing for the modelling of the process behaviour. Each process net is named by the author a module net. From such separate module nets, to capture the composite business processes workflow the authors use the fusion of similar input/output places. They also add an extra transition at the beginning to allow running composite modules in parallel. Similarly, to get a unique final state from the composition they add an extra transition at the end relating all final places from each process.

On the basis of this module and composite module nets, the authors introduce two properties called compatibility and usability. The *compatibility* property allows checking whether another service module net (named here environment) is composable with a given module net. In other words, for each input place there should correspond an output place (from the environment module); otherwise the composition could not take place (i.e. incompatibility). The *usability* checks whether the composite module net is weak (i.e. each initiated process comes to a proper final state). Although the authors mention that composite module nets could be translated to WSBPEL, we argue that it is no obvious task because input/output places fusion could not be regarded as invocation or receive messages. Besides that simple place/transition nets do not consider complex data modelling such as message arguments and advanced business rules. Subsequently, we will abbreviate this model as BPW-PN (i.e. *B*usiness *P*rocess Web*S*ervice with *P*etri *N*ets).

Place/transition Petri Nets-based model for Web services [34] proposes an expressive net-based model that allows capturing in a declarative way different service combinations and their respective specificities. Moreover, the proposed algebra caters for the creation of dynamic and transient relationships among services.

Authors define first what they called *Services Net*, which is just a P/T Nets with one starting place (without input arcs) and one final output place (without output arcs) and labels (as operation names) associated to arcs. On the basis of this Service net notion, they define Web services as a tuple (`NameS; Desc; Loc; URL;CS; SN`) with `Names` as the service name, `Desc` as the description of the service functionalities, `Loc` and `URL` for services location and URL, `CS` as the name of the service components (if being composite composed) and finally `SN` as the service behaviour expressed in terms of a service net.

With this Web service description as tuple, they built a rich algebra that combines several Web services in different ways (sequence $[S_1 \bigodot S2]$, alternative choice $[S_1 \bigoplus S2]$, arbitrary sequence $[S_1 \Diamond S2]$, iteration μS, parallelism with communication $[S_1 \|_{\mathcal{C}} S2]$, discriminator operator $[(S_1|S2) \rightsquigarrow S_3]$, refinement $[(Ref(S_1; a; S_2)]$, selection $[S_1(p_1, q_1) : S_n(p_n, q_n)]]$).

Using usual Petri Net's flow capabilities, they semantically interpret each of these syntactical constructs. The most noticeable effort was about the interpretation of the selection, where a specific part of the service should concern the request for a selection.

As they have opted for simple Petri nets, techniques for analysis of different properties such as deadlock or liveness can be achieved using techniques like place/transition invariants and siphon/trap in Petri nets. Also, they suggest the use of bi-simulation techniques.

Although the proposed algebra and the contribution as a whole is very rich and significant, we may point out some limitations related to the practicability of the approach. The first shortcoming is the lack of relating this work with the capabilities available Web service technology such as BPEL and WSDL. Indeed, on the one hand, the proposed algebra seems to be highly expressive that this technology in terms of its richness in operators. That is, apart from the usual operators such as sequence and parallelism, no available Web-based language uses the so-called advanced operators like arbitrary sequence, refinement or parallelism with communication. However, on the other hand what is clearly missing is the handling of data, such as message parameters, conditions, etc which are among the main keystones of Web service languages like BPEL. Subsequently, we will abbreviate this model as PN4WS.

It is worth mentioning that very recently, the authors of this proposal have partially tackled the adaptability. In fact, in the extension called self-adapting recovery net [35], they propose how to dynamically handle exceptions, errors and dynamic changes of the structure of the net (i.e. by dynamically deleting, adding and removing places and transitions). What is out of scope in this adaptability is the runtime modification of existing arc-inscriptions and conditions (governing business rules) as we are intending to achieve in the coming chapters. Indeed, the structure of a given Petri nets can be dynamically updated without the business rules changing: A fact that we aim to overcome.

[64] proposes a specific form of place/transition net for specifying business to business (B2B) composition. Instead of usual Web services terminology, the authors used the equivalent concept of *E-service*. An E-service is specified

both in its static interfaces and in its behaviour. Specifically, an E-service communicates through messages, including both the ones E-service receives and the messages it produces. So it is more close to BPEL description but with reactive behaviour. In this approach, each E-service is formally specified by the so-celled *E-service net*, which is a Petri nets with three categories of places. Input messages are drawn using rectangles as places. Output messages are drawn using bold rectangles. Any of the places for capturing the internal behaviour are modelled as usual circles and named control places. Transitions model the behaviour using these places as input/output.

Using this E-service net construction, the authors then proposed the notion of E-service orchestration net. This form of net allows composing of several E-services net, which is a specific net connecting at least two E-service nets, and specifying the routing of messages and the act of passing the task of the orchestration from one organisation to another. For such composition they introduce a new type of places called orchestration places and is drawn as hexagonal. Such orchestration places allow indication of the current organisation performing the different E-service composition, which may change as the composition goes on over time.

This E-service orchestration model provides thus a mechanism for supporting control of E-services process evolution in terms both of control and data flows, and for distributing and assigning process responsibilities. To enhance the practicability of their approach, the authors show how a significant part of an E-government could be specified using an E-service orchestration net. They also mention the application of usual analysis techniques such as deadlock freeness of the overall process and reachability of the final configuration of the involved E-services, which can be verified by analysing the configuration graph of the net.

This work is very interesting from a practical point of view, as it permits to easily understand E-service behaviour and its orchestration (e.g. specific place form of input/output messages, organisations, etc). However, one of the missing important issues is the relationship to current web service composition languages such as BPEL and WSCI. The approach also uses just black-dot tokens without advanced data structure as the later web-languages (BPEL, WSCI, etc) require. Subsequently, we will abbreviate this model as E-SvPN.

3.2.2 Web Services Models Based on High-Level Petri Nets

This approach [70] aims at enriching the semantical capabilities of semantic web languages such as DAML-S and DAML-OIL. For that purpose, the authors first adopt the situation calculus for enriching conditions and effects using the rules of this calculus. Situation calculus are more or less similar to first-order logic with modality operations like possibilities.

Having expressed such advanced first-order formulas using an extension of DAML-S, the authors propose to interpret them using Petri nets. The benefits here are to graphically animate and do some property analysis of this enriched semantic web language.

The translation is very intuitive and could be highlighted as follows. Each formula captures a specific transition. Input place types of such transitions correspond to different atomic processes or predicates. The post-conditions are captured as output places. From these basic transitions, different forms of service flow can be easily modeled using Petri nets. Such flow includes sequence, parallel, if-then-else, etc.

Although situation calculus is very rich, it remains still very hard to be understood by non-experts. This makes the translation to Petri nets more harder. Subsequently, we will abbreviate this model as SmWbPN

[69] proposes to model Web-engineering by adopting the strengths of the high-level Petri nets variant based on Nets in Nets. This variant allows tokens from places to be themselves place/transition nets. That means by firing a transition, a new instance of a net could be created as output inscriptions and another net is destroyed from input places.

The authors exploit these advantages for modelling Web services. They put forward a four-layered refinements based approach. First, giving a complex web-application composed of several interconnected Web services, they conceive it as a net called service network. Each place of this (network) Petri net is then itself conceived as a Petri net called Web service container. This corresponding net allows managing the creation and deletion of service instances of this type. Places from this service container Petri net are then at their turn regarded as a net called Web service. This net allows for requesting/responding to different external invocations and for possibilities of delegating tasks to other Web services. Finally, some places in this net correspond the flow of elementary operations reflecting the proper behaviour of such service.

Important to emphasise here is that this four-layerd based approach to web-engineering modelling is inspired by the authors previous work based specifying multi-agents using Nets in Nets approach. This approach allows more flexibility and adaptation, besides separation of concerns. That is, the composite Web service of abstractly conceived, then its details (service contain, behaviour, etc.) are further specified in incremental way.

The philosophy of nets in nets approach still remains very hard to understand, let alone apply it in complex domains such as Web-engineering. Moreover, although the authors mention that their work could be easily combined with existing Web service languages such as BPEL4WS, it seems to be a non-obvious task. Indeed, as BPEL is based on a one layer-based methodology, where all messages and conditions and their flow are explicitly described, the proposed approach makes it very hard to obtain this whole and global model in a tractable. Besides that, the inter-relations between messages from different Web services as BPEL explicitly describes become impossible to capture following this layered approach. In other words, the composition as understood in Web service technology is missing. Subsequently, we will abbreviate this model as 2NetsWS

In [99, 100] very promising approach for Web service modelling has been recently put forward. This approach is based on Coloured Petri Nets [42], one of the mostly accepted and widely adopted variants of High-level Petri Nets both in academia and industry. Coloured Petri nets enhance standard Petri nets with the primitives for the definition of the data types and the manipulations of data values. Moreover, CPNets propose advanced structuring mechanisms including Place/Transition fusion and Nets hierarchy.

This CP-Nets-based approach for Web services aims to achieve at least three objectives:

- The *composition* of Web services incorporating partners with complex conversation protocols, and;

- The automatic derivation of *conversation* protocols from the composition for each involved Web service;

- Validation/Verification of a Web services composition and its conformance to component services conversation protocols.

The Web service composition proposed in this approach is an orchestration one, specifically based on a form of reactive *stateful* BPEL4WS. This extension of reactivity to BPEL is very important as most real-size service composition are long-term transactions with instance playing different roles. More precisely, the process aspect of a Web services composition specified in BPEL4WS can be represented with CP-nets. This CP-nets-based process composition model is defined as follows:

- The process of the composite service is represented by a CP-net (denoted by `NetS`); each partner is represented with the CP-net model for its conversation protocol (denoted `NetP`); `NetS` interacts with `NetP` through arcs connecting the *in*- and *out*-places of `NetP`. Each arc must be labeled with a token variable that matches the coloured set declared for the in-place/out-place.

- Messages (events) and process variables are represented by tokens. Since the concrete content of the messages (variables) is not known at design time, abstract colour sets are declared for the messages and variables. Therefore, each colour set is kept small to speed up the analysis.

- A BPEL4WS activity is usually mapped to a CP-nets transition. A `<receive>` activity is represented by a transition which has an *in*-place. A `<reply>` activity is represented by a transition which has an *out*-place. An `<invoke>` activity is represented by a pair of transitions, one of them may fire a request token to `NetP`, and the other may wait for a token from `NetP`. A structured activity is represented by a substitution transition. The control flow between activities is captured by connecting the activity-related transitions with arcs, places, and transitions purely used for control flow purpose. More refined control flow can be expressed with arc inscriptions and transition guard expressions.

- At this time, certain aspects of the composite service are ignored, such as compensation handling, fault handling, and message correlation.

This composition is illustrated with a travel agency example. For instance, airline operations like `CheckSeat`, `ReserveSeat`, `BookSeat` or `CancelSeat` should be logically `ordered` and more importantly synchronised or triggered by corresponding `TravelAgency` operations such as `FindBestIterinary` and `BuildIterinary`. These `Agency-AirLines` operations have also to be synchronised with the customer operations such as `TripOrder`, `ReserveReq`, `BookReq` and `CancelReq`. The synchronisation is captured by adequate transitions. The order between different operations within each service (as conversation model) are modelled using appropriate transitions and places.

The CPNets proposal for WS allows also for conceiving the conversation protocol as a WSPNet, where:

- Each operation is represented by a transition. An input Place (if exists) connects to the transition, and represents the reception and buffering of inbound messages for the operation. The transition also connects to an output Place (if exists), which represents buffering and transmission of outbound messages for the operation.

- Each WSDL operation is represented by a CP-net transition. The transition also has one input place which stands for the pre-condition of the operation, and one output place which stands for the post-condition of the operation.

- Messages exchanged by the service and its clients are modeled by tokens. Small-sized colour sets are used to capture the protocol-relevant feature of a message.

- The synchronisation rules of the conversation protocol are captured by connecting the transitions (each of them representing an operation) with places, arcs, and dummy transitions used only for control flow purposes.

This so-called service conversation model is automatically generated from the composition, using a deterministic algorithm that the authors propose.

As we just emphasised, this approach is very expressive and more close to the current investigations within Web services such as reactive complex composition and rich conversation models. Nevertheless, we may point out the following crucial shortcomings that we aim at overcoming using our approach. Firstly, although CP-Nets is a very expressive formal framework, the fact that the authors where completely bounded with the limited capabilities of composite BPEL, business rules governing different operations are completely missing. This is a very severe drawback as business rules are omnipresent and may change over time, and should be present in the modelling of service composition. Secondly, the fact of associating with each operation in several places (and several transitions), the modelled service composition could easily become untraceable and confusing (place explosion). In this respect, advanced CP-Nets structuring mechanisms such as place/transition fusion and hierarchy could be very helpful. Thirdly, the proposed Web service composition model tackles just the orchestration. That is, the interaction between Web services as a choreographical manner is completely missing. Subsequently, we will abbreviate this model as CpN-WS

3.2.3 High-Level Petri Nets for WS-Adaptivity

Petri nets are among the leading specification frameworks for complex distributed systems. They enjoin several determinant characteristics.

- They introduce few concepts such as places for holding system states and transitions for capturing system functionalities such as actions, operations and their behaviour.
- They are graphical promoting more understandability (for non academic) and allowing system animation through the tokens game.
- They are mathematically founded, with the possibilities of different semantics (e.g. interleaving, steps, concurrency) depending on the specificities of modelled applications.
- They allow system analysis to check properties such as deadlocks, liveness, safety, etc. Among the well established analysis techniques we may cite Place- and Transition-invariants to siphons and traps.
- With the development of High-level Petri nets, different structured complex data can be dealt with leading to the specification/animation and verification of real-size complex applications.

All these qualities have made High-Level Petri nets one of the prominent candidates for specifying and validating and analysing Web services applications. Confirmation of the claim is the growing interest and numbers of approaches put forward for specifying Web services as we have seen in the previous sections. In particular, the benefits of adopting High-Level Petri Nets for Web services modelling may recall the following.

- As Web services are by nature *distributed* applications, the concurrency behaviour that characterises High-Level Petri nets put them among the most suitable conceptual framework for Web services

- As Web services composition is driven mainly by business processes, the natural ability of High-Level Petri nets in capturing different activities ordering (e.g. parallel, sequential, and-join, or-join, choice, etc.) promotes their modelling.

- The explicit states in terms of places in High-Level Petri nets enhance the specification of *reactive* Web services applications, which are omnipresent and represent one of the challenging classes of Web services to specify and reason about. Indeed, reactive Web services are *stateful* which require conceptual models able to combine composition and conversation features.

- The advanced structuring mechanisms (such as the object-oriented ones with inheritance, composition, aggregations), that propose some variants of Object-Petri nets (see the reviewed models), permit expressing complex constraints and conditions about the Web services behaviour. These constraints can easily be expressed in terms of business rules and be changed semi-automatically (using specific patterns).

Nevertheless, as we demonstrated in our extensive survey about current High-Level Petri nets-based Web services modelling techniques, not all potentials of the Petri nets-based formalisms have been exploited and more investigation is required to cope with Web services adaptivity and agility.

However adaptivity with respect to workflows have been largely explored using different extensions to Petri nets. Although workflows in contrast to Web services are mostly *centralised*, ideas from such proposals could always be helpful to understand solutions to adaptivity in business processes in general. In this respect two approaches remain dominant. The first approach was introduced by Van Aalst in [95, 2]. This approach is based on a particular class of ordinary Petri nets called Workflow-Nets, and it uses object-oriented behavioural inheritance as the main vehicle to cope with adaptivity. The second recently introduced in [76] is based also on simple Petri nets but it adopts the concepts of region and recovery principles to achieve adaptivity.

Due to lack of extensions of such approaches to adaptivity in Web services, and as we motivated in the introduction of this report the objective of this thesis is to deeply foster all potentialities of High-level Petri nets for coming up with an adequate conceptual model that promotes automatic adaptivity in reactive distributed Web services.

For that ultimate objective we are endeavoring to achieve the following. Firstly, we aim in promoting object-oriented mechanisms with concepts from componentisation (e.g. explicit interfaces, separation of concerns, modularity, etc) to express business rules in High-level Petri nets in a satisfactory way. This will represent the first milestone towards adaptivity. Secondly, we aim in relating the proposed conceptual model with UML-based diagrams and this to ease the modelling and pinpoint the volatile parts (business rules) in a straightforward way. Thirdly, to fully manipulate such business rules at runtime we plan to adopt reflection techniques [63, 18].

3.3 A Comparison of Web Service Models Based on Petri Nets

3.3.1 Web Service Criteria for Comparison

Recapitulating from our investigations on many issues related to the modelling of Web services, in the following we introduce a list of fairly exhaustive criteria for assessing the adequacy of any conceptual model as foundation for Web services. For sake of clarity, these criterion are classified into five categories: *Composition, Service modelling, Practicability/Expressiveness and Adaptability* aspects. In the following, first we explain in general the purpose of each category, and then for each one we propose a set of criterions that we argue as required.

Composition

As widely recognised, composition is the main essence in Web service and service-oriented architecture. Composition is the ability to bring together the functionalities of more than one service (interfaces) to build complex service and achieve the expected complex customer requirements and demands. Generally, Web services Composition tackles inter-organisational services.

- Composition criteria:

  Choreography/Orchestration: This criterion means the explicit ability to distinguish between orchestration and choreography. Orchestration means the ability of specifying a specific service that interacts (i.e. be composed) with other services through messages invocation, reception and replication. This part of restricted single-view composition is promoted by languages like BPEL4WS, DAML-S. Also, orchestration deals with the business flow of services composition. Choreography in contrast allows interacting or composing several services through their interfaces and in a balanced way.

Stateful Orchestration: The ability to conceive the services orchestration instances in a persistent and reactive way.

Stateful Choreography: The ability to conceive the services choreography instances in a persistent and reactive way.

Stateful Conversation: To allow composition of complex web-applications, service interfaces (e.g. invoke, receive)is not sufficient; instead a complex conversation is mostly required between different invoked and received operation services. Such conversation should also be modelled in a stateful way to cope with different instances and their states.

Dynamic Composition: Although Web services aims to generate composite services in a runtime way, existing Web services languages still achieve it statically and manually. As pointed out in [72] business rules may significantly contribute to make such composition more dynamic.

Stepwise Composition: Web services composition involves several activities including: interface definition, interface behaviour specification, orchestration, conversation and choreography. To master this complexity a clear stepwise methodology is required to optimally organise and coordinate such activities.

Service Modelling

Under this crucial category, we refer to the capabilities of the adopted conceptual model for specifying service behaviour satisfactorily. This includes, for instance, the different abstraction mechanisms for coping with service complexity, the capability for dealing with stateful specification, etc.

- Service Modelling:

  Abstract/Concrete Interfacing: Differentiating abstract interface description from a concrete one is a very beneficial. Indeed, this separation allows at the abstract level to concentrate more on the main functionalities of the service, whereas at the concrete level more detail including qualities of of selected service, its location, etc have to be considered.

  Service Interface Behaviour: It is the ability to capture the behaviour of the abstract service description in an expressive and stateful way.

  Concrete Service Behaviour: We argue that a good conceptual model should also allow modelling the behaviour of a concrete service with all its qualities of services and implementation details.

  Service Data Abstraction: In modelling abstract or concrete (composite) services, the ability of explicitly dealing with data such as (message parameters, operation conditions, etc) is very crucial for a conceptual model to be acceptable.

  Service Data Structuring: To cope with complex data-intensive services, more advanced structuring mechanisms are required such as inheritance (i.e. service classes and subclasses) and service aggregation.

Service State Intra-concurrency: First, we argue that an explicit modelling of each *service state* in the service behaviour specification or in its composition is very relevant. Besides that, the ability of concurrently applying more than one operation on different parts of such state enhance the service performance.

Service Inter-concurrency: Concurrency should also be supported between different service state instances.

Practicability/Expressiveness

Formal methods of any sort remains still very hard to use, in complex applications such as Web services, even those based on Petri nets. To enhance such practicability, we argue that associated criteria should be set. These criterions include, for instance, the capabilities of: stepwise construction of model, methodological support with semi-formal diagrammatical models such as UML, and hiding the formal technicalities and semantics.

- Practicability/Expressiveness:

Expressiveness: As we just mentioned an adequate conceptual model for Web services has to allow explicitly specifying service data, messages, states, etc.

Compactiveness: To absorb the huge complexity of real-size Web services at the modelling phase, the model has to be compact enough to represent complex web-applications. Refinement steps may be necessary to incrementally deal with all service details. With respect to Petri nets, Place/transition-based models could easily lead to place/transition explosion. Even high-level Petri nets need more structuring mechanisms to cope with service complexity.

Relationship to Current WS Technology: As XML-based Web services languages such as (BPEL, WSCI, WSDL, ect). are becoming the de-facto standards, any adequate conceptual model should be able to intuitively and automatically be translated into such languages. Also, foundation and enrichment capabilities of such languages is very important.

Semi-formal Diagrammatical support: As diagrammatical-based informal methods like UML is gaining more and more acceptance in software-engineering, we argue that a suitable conceptual model for Web services should include some UML-diagrams in its earlier modelling phases.

Hiding of Formal Semantics: The previous criterion is an important step towards hiding formal details when clear translation steps are proposed to automatically generate the conceptual models from its UML-diagrammatical descriptions

Tools for Validation/Properties Analysis: Tools are a determinant factor to enhance the practicability of any conceptual model for Web services. It allows generating rapid-prototyping for validation purpose and the verification of essential properties of the system.

Adaptability

Another urgent aim of Web services is to achieve a dynamic adaptation of composite services. Requirements on conceptual models to be adaptable include the following criterions: the ability to explicitly conceive business rules as the main volatile part in any (composite) service; the ability of reasoning and dynamically changing its behaviour through reflection capabilities.

- WS Adaptability:

  Business Rules Modelling: Business rules are the most volatile part of any organisation [49]. It describes the functionalities/policies and rules for doing business. The ability of explicitly modelling such business rules is therefore determinant for making services adaptable.

  Runtime Adaptability: This ability implies explicitly separating between the adaptability-level and the conceptual "base-level" model. This allows shifting up/down at runtime any emerging business from the adaptability-level to the conceptual model, and thus adapting and evolving it as needed. In the literature it is referred to as reflection techniques.

Comparison and Comments

After sketching the main features of the each of the recently proposed formal Petri Nets-based conceptual models to Web services, that we followed by a rich set of criteria that we argue should characterise any widely-accepted formalisms for Web services, we assess these criteria with respect to the features of each of the above-described models for Web services, that are based on different variants of Petri nets. The table below summarises the result of each approach with respect to the afore-described criteria categories. The used legends are: '$\surd$' standing for Yes (i.e. the criterion is supported by the model) ; '$\times$' standing for No (i.e. the criterion is not supported by the model) and '$+/-$' for a non satisfactory fulfillment of the criterion (i.e. the criterion is only partially supported by the model).

	BPW-PN	PN4WS	E-SvPN	SmWbPN	2NetsWS	CpN-WS
Composition Criteria						
Choreography/orchestration	×	×	+/−	×	×	×
Stateful Orchestration	+/−	+/−	+/−	+/−	√	√
Stateful Choreography	×	×	×	+/−	×	×
Stateful Conversation	+/−	+/−	+/−	×	+/−	√
Dynamic composition	×	+/−	×	×	+/−	+/−
Stepwise composition	×	×	×	×	+/−	+/−
Service modelling						
Abstract/concrete interfacing	×	×	×	×	×	×
Service interface behaviour	+/−	+/−	+/−	√	√	√
Concrete Service behaviour	×	×	×	×	×	×
Service data abstraction	×	×	×	√	√	√
Service data structuring	×	×	×	×	×	×
Service state intra-concurrency	×	×	×	×	×	×
Service inter-concurrency	+/−	√	√	√	√	√
Practicability/Expressive.						
Expressiveness	+/−	×	×	+/−	√	√
Compactiveness	×	×	+/−	+/−	√	+/−
Relationship to WS technology	+/−	×	×	+/−	+/−	√
Diagrammatical support	×	×	×	×	×	×
Hiding of formal semantics	+/−	×	+/−	+/−	+/−	+/−
Tools for validation/verification	+/−	×	+/−	+/−	√	√
WS Adaptability						
Runtime adaptability	×	+/−	×	×	×	×
Business rules modelling	×	×	×	×	×	×

Table 3.1: Comparison of (High-Level) Petri Net-Based Models for Web services

As depicted in this comparative table, it can be easily notice that all conceptual models fail in coping with adaptivity. Apart from these shortcomings for all models, the other criteria are supported by some and absent from others. Besides that, as expected High-level Petri nets-based approach are more expressive and respond positively to more criteria than those based on simple Place/Transition nets. The objective of setting criteria that are not supported by any of these conceptual models is to prepare the path toward the model we are working on, and which will cope positively with all the criteria including adaptivity.

3.4 Summary

This chapter brings the reader closer to the recently developed approach for formalising Web services using high-level Petri nets. Moreover, after summarising such recent petri nets and High Level Petri Nets based proposals, a set of criteria has been put forward for classifying and assessing such proposals. This comparison will allow us in particular to pinpoint different shortcomings of these proposals with respect with what we are aiming for, namely business rule-centricity, harmonious complementarity between orchestration and choreography and runtime adaptability.

Chapter 4

Service-based Petri Nets: Foundation and Methodology

In this first main chapter, we are putting forward an innovative foundation endowed with a supporting methodology for rigorously modeling and validating distributed, knowledge-intensive, adaptive service-driven applications. More precisely, at the foundation-level, as we pointed out in the introductory section, a new variant of high-level Petri nets will be motivated and progressively, rigorously defined and illustrated with the non-trivial agency case-study. That is, instead of coping with just structural aspects in defining service interfaces and statically composing them in purely process-centric manner as currently the case with different XML-Based Web standards.The variant of Service-driven High-Level Petri Nets (we refer to as CSrv-Nets) we are proposing allows for intrinsically addressing *behavioural* features in a rule-centric, concurrent, adaptive and compositional manner.

On the methodological side, and because formalisms such as high-level Petri nets still remain hardly understandable and accepted by (cross-)organisation stakeholders (e.g. managers, users, customers and even programmers), we are going to promote the practicability and the wide-usability through the early adoption of semi-formal diagrammatical and standardised artifacts both for structural and behavioural features in service-driven applications. More precisely, all structural features of service-driven business applications are first described using stereo-typed UML 2.0 use-cases and class-diagrams. Behavioural aspects are captured through event-driven business rules, which are inherently understandable, evolving and process-independent. Only after such widely acceptable and accessible semi-formal descriptions, of any service-driven application at hand, we then forward a smooth and semi-automatic shifting towards the proposed rigorous service-driven Petri Nets formalism. In the next section, we present further motivations and wider insights for both the forwarded formalism and its supporting methodology for developing complex adaptive and rule-intensive service-driven applications. In the second section, we develop on the earlier preparatory semi-formal phases for capturing the structural and behavioural features in service applications, namely the UML2.0 enriched use-cases and class diagrams as well as the event-driven business rules, respectively. Since such artifacts and mechanisms are widely-known, we concentrate on their application to the running case study. In the third and first main section within this chapter, we progressively motivate and rigorously define the structural features of our ”concurrent” Service-driven High-Level Petri Nets (CSrv-Nets) formalism. We further present how all the defined CSrv-Nets concepts are incrementally and smoothly derived from the semi-formal phase, and illustrate them using the Airline service. In the second main section, we focus on leveraging such CSrv-Nets structural features with corresponding behavioural ones on the basis of agile business rules. In the fifth section we briefly report on possible operational semantics for validating the resulting CSrv-Nets-based conceptual model, which may boost any algorithmic graphical animation. Finally, to demonstrate the practicability of the approach and how it can cope with any complex service applications, we address the other services related to the Agency case-study, that is, in addition to tackling the Airline service we apply the approach to the other services, namely hotels, customer and banking services.

4.1 Motivation on the Conceptual Framework and Methodology

The purpose of this section is twofold. Firstly, we revisit the arguments and the potential of opting for leveraging high-level Petri nets to formalise, validate and reason about adaptive distributed knowledge-intensive service-driven applications. Secondly, we review the different incremental steps of the proposed methodology, we are bounded to while developing such adaptive service-driven applications.

4.1.1 Potentials and Shortcomings of High-Level Petri Nets

As we reported in the previous chapter, several ongoing approaches are being proposed for formalising and reasoning about service-driven applications and Web services in particular. In the following we first enumerate some of the main advantages and arguments in favor of our decision of opting for High-level Petri nets. As achieved in chapter 3, we emphasised some of the main shortcomings of existing similar Petri nets-based proposals for service applications. These argumentations will allow us to pave the way for leveraging high-level Petri nets towards overcoming such limitations , and result thereby in the new variant we are proposing in this chapter referred to as CSrv-Nets.

Understandability via Visualisation: Experience shows that formalisms endowed with graphical descriptions are more accepted by cross-organisation's stake-holders (not just designers and programmers). Understandability is further essential in bridging the gap with the business-level, where first, intuitive description of the service-driven application, at hand, is given in terms of global goals and processes.

Concurrent and Distributed Behaviour: As we pointed out distribution is of one of main characteristics of (advanced composite) services. Consequently, the targeted formalism requires to intrinsically support concurrency and distribution.

Type- and Instance-level Support: We argue that one of the shortcomings of Web standards, such as WSDL, BPEL and WSCI, are their inability to intrinsically cope with the instance-level, where one may directly address and reason about specific services. Moreover, coping with both the type- and instance-level represents thus a critical requirement to tackle service states, and thereby explicitly dealing with persistency and conversation.

Validation and Verification: As we are going to bridge the gap between the business-level and the conceptual level, requirements validation are thus essential. The validation should thus include: requirements missing, misconception, misunderstanding, conflicts and so on. Graphical validation in this respect is highly requested. Besides such validation, the formal verification of crucial properties are possible.

Advanced Abstraction Mechanisms: On the one hand, we argue that current monolithic interfacing are not sufficient to offer features to different customers and/or focussed compositions. Object-oriented advanced mechanisms such as inheritance and aggregation are thus deemed necessary to be supported by serious candidate formalism. On the other hand, the ability of addressing componentisation and explicit inter-component interactions represent prerequisites to cope with service compositions via (structurally and behaviourally) rich interfaces.

Despite these important potentials, we argue and demonstrate in this chapter that at least the following additional features must be soundly integrated in any traditional high-level Petri Nets variant.

Explicit Representation of Service States: To cope not only with short static and stateless service interactions, we propose to promote long-running service interactions with conversational and thus stateful features. For that purpose, an advanced and flexible form of service state is proposed for our CSrv-Nets.

Handling of Event-driven Business Rules: This has the most potential and benefit of the proposed CSrv-Nets variant. Indeed, to our best knowledge no existing high-level Petri nets variant can explicitly and intrinsically deal with business rules, which are essential when it comes to adaptability and evolution. We thus propose how to inherently and soundly integrate event-driven business rules within our CSrv-Nets formalism.

Separation Between Imported/Exported Messages and Event Triggering: Though some existing high-level Petri nets differentiate between input/output places, no existing similar formalism devoted to service-driven applications allow for explicitly and semantically separating between imported/exported (that is, requested/offered messages). Besides achieving such distinction, we also capture the notion of triggering event to reflect event-driven business rules.

Incremental Behaviour from Structural Features and Business Rules: Most of existing high-level proposals do not support the designer for systematically constructing the behavioural features of the net, which leave too much confusion and room for different behavioural interpretations. We circumvent this serious problem by supporting the designer with explicit clear steps on how to conceive the behavioural issues, i.e. net places and transitions and their inscriptions, from the already defined structural aspects and related business rules.

4.1.2 Stepwise Supporting Methodology

As illustrated in Figure 4.1, the proposed approach is methodologically composed of the following phases.

- Phase One :

UML/Business-rules Requirements Phase: In this preliminary phase, the (English) informal description of the business-driven application at hand is semi-formally and diagrammatically expressed in terms of UML Use-Cases and Class-diagrams. Besides that, all related intra- and inter-organisational business rules that come from business people(manger,

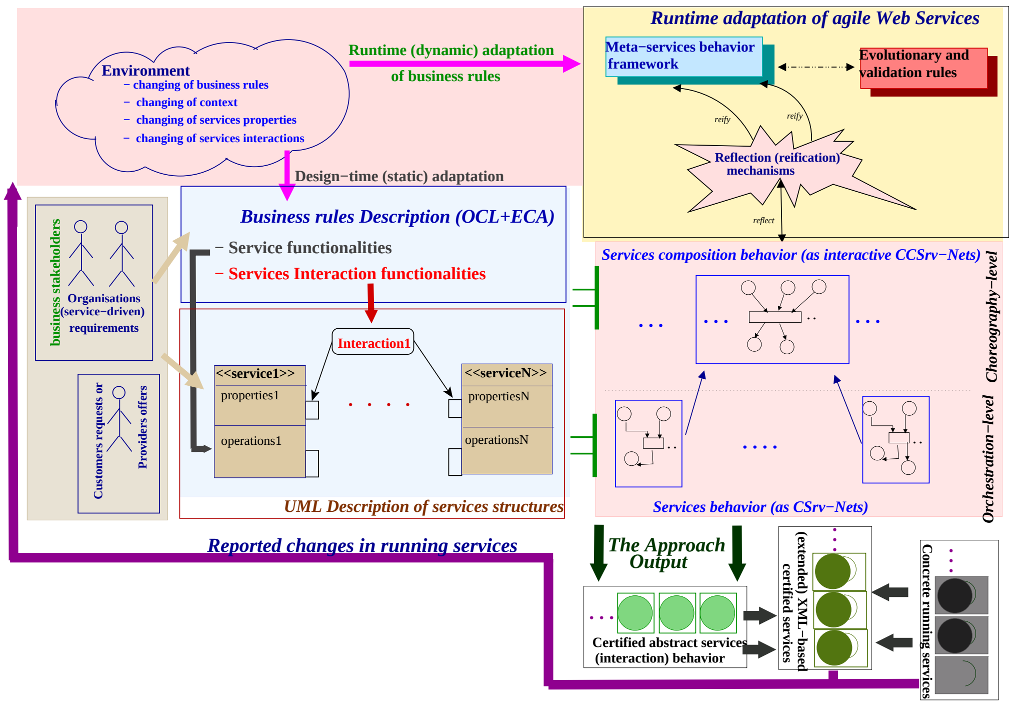

Environment
– changing of business rules
– changing of context
– changing of services properties
– changing of services interactions
Runtime (dynamic) adaptation of business rules
Runtime adaptation of agile Web Services
Meta–services behavior framework
Evolutionary and validation rules
reify
reify
Reflection (reification) mechanisms
reflect
Design–time (static) adaptation
Business rules Description (OCL+ECA)
– Service functionalities
– Services Interaction functionalities
business stakeholders
Organisations (service–driven) requirements
Customers requests or Providers offers
Interaction1
<<service1>>
properties1
operations1
<<serviceN>>
propertiesN
operationsN
UML Description of services structures
Services composition behavior (as interactive CCSrv–Nets)
Choreography–level
Orchestration–level
Services behavior (as CSrv–Nets)
Reported changes in running services
The Approach Output
Certified abstract services (interaction) behavior
(extended) XML–based certified services
Concrete running services

Figure 4.1: A disciplined approach for service rule-centric adaptive business applications.

stakeholder, etc) governing the behavioural features of different basic and composite services are to be clearly described, following in particular the well-known Event-Condition-Action (ECA)paradigm which can be integrated in the CSRV-NETS as a transition. The derivation from Phase one to Phase two can be achieved informally as follows:

- The places of the net are precisely defined by associating with each service message generator from UML class diagram one 'message' place.
- With each service state sort a 'state' place is associated.
- Transitions, which may include conditions (business rules), reflect

the effect of messages on service states to which they are addressed.

- Phase Two:

Concurrent Services Nets Specification/Validation Phase: This phase is decisive as it allows to define in a precise and concise way all functionalities and behaviours of different service components and their interaction (i.e. service interfaces, elementary and composite services) and validate them against misconception, misunderstandings, conceptual mistakes, etc. For this crucial phase we are proposing a variant of high-level Petri nets, that reflects all structural and behavioural features of elementary or composite services, such as distribution, persistency (stateful), conversation. Moreover, we are enhancing the practicability of this conceptual model we refer to as CSrv-Nets, by hiding as much as possible its tedious mathematical aspects.

- Phase Three :

Adaptive Service Nets for Runtime Evolution: This phase allows us to endow the conceptual model CSrv-Nets with a an adaptability-level so that runtime evolution and adaptivity of different services behaviour can be achieved at run time and in a consistent and incremental manner.

The purpose of this chapter is to develop on the first two phases, with a special emphasis on the second crucial and decisive rigorous conceptual phase. More precisely, the following will be developed:

(1) We formally define the main structural ingredients of the variant of Concurrent Service-Driven High-Level Petri Nets (CSrv-Nets) we are

proposing. That is, in the first part we present how structural aspects of service interfaces including service states and messages are formally specified, and how they can be directly derived from UML Use-Cases and "Service"-diagrams (regarded as stereo-typed class-diagrams).

(2) We then define the behavioural aspects of service interfaces using CSrv-Nets, and we demonstrate how (intra-organisational) business rules can contribute to the construction of this concurrent and statefull conversational behaviour (unlike static stateless XML-languages like BPEL for orchestration).

(3) The approach is further applied to the non-trivial version of the agency case-study.

4.2 The Semi-Formal Phase Applied to the Agency Application

This section presents an informal description of a simplified form of a realistic case study dealing with travel agency systems, which is a typical illustration in Web services.

4.2.1 Travel Agency: Informal Description

In analysing a medium-size case-study, we are considering in this thesis the widely adopted *travel-agency* service-driven application. Indeed, this application at least at the research level, is considered as one of the benchmarks for assessing the capabilities of any proposal.

In its simplistic variant, this case-study could be formulated as follows. User or customer, may log in to any travel agency service, and formulate his/her request for simple travel/accommodation or a complete sophisticated package (e.g. travel, accommodation, attractions, car rental, sight seeing, etc). The agency then contracts the related services including the payment institution (e.g. banking, credit-card). A conservation business process will be created between all these services and the customer, can in the middle cancel such request (under business rules).

Most of the service-oriented research tackling this case-study (or any others like auctions, E-shopping) restrict their investigations to the modelling (and partly verification/validation) of pure *process-centric* compositions. That is, for instance, first the request from the customer is received, their the agency accordingly dispatch messages to associated services (e.g. airlines, hotels, attractions, etc.). Once receiving a reply from such services, the customer is again approached for accepting/rejecting the proposed offers. Once confirmed, flight tickets, rooms reservations and so on will be booked and payment via credit card or bank transfer will be performed. At this level, if the customer asks to cancel, he must pay a penalty depending on different regulations and so on.

In other words, what has been so far tackled is mostly how to model and reason about rigid and process-centric service compositions. As we pointed out in our motivation, realistic composite services are more complex and regarded to be highly flexible, and thus go far beyond such simplistic rigid and process-centricity. More concretely, with respect to the travel agency case-study, we aim to tackle the following aspects. Firstly, the agency being a service for airlines, accommodation or any other involved service, they are all governed by knowledge behaviour-intensive regulations, that is, by policies or simply business rules.

Moreover, to stay competitive and increasingly attract more customers, such business rules are designed to such customers and more importantly are rapidly and unpredictably evolving and changing to face fierce market globalisation. For instance, airlines propose reductions on the basis of number of persons, their ages, their destinations, durations in case of two-way tickets, the seasons, and so on. Ignoring or postponing such rules till the deployment, means simply putting a huge gap between the realistic world and automated services (which become more than impractical). Besides such service-focussed business rules, there are those crossing-(organisational) services, which are decisive while composing complex services. For instance, the agency through a negotiation with its partners (e.g. specific airlines, hotels, etc) proposes different vacation packages governed by attractive business rules.

4.2.2 Travel Agency : UML Diagrams and Business Rules

To illustrate this simplified variant of vacation arrangement, Figure 4.3 presents its corresponding use case diagram.

Travel Agency Scenario:

- The travel agency provides an online system for customers that offer the widest possible range of vacation packages depending on environmental situations (i.e seasons, events, year, short/long vacation, etc) as well as customers preferences and situations (i.e. individual/group, complete/partial packages, etc.)
- Service providers (flights, accommodations) publish their services on the repository by making them available using current Web services.

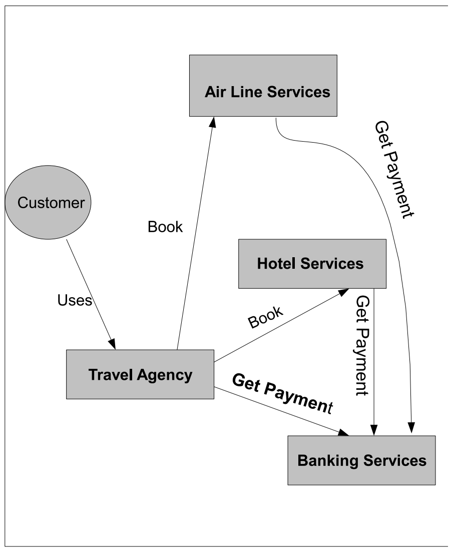


Figure 4.2: A Overview of the Travel Agency UML Use-Case

- Banking services (i.e. credit cards services) enable customers to use their credit cards to make payments via Web services.

All operations between services are perform automated without need for a human intervention. However, only customer in this case are humans being.

Steps for the scenario can be summarised as follows:

(1) The user enters the targeted URL address of a travel agency service.

(2) The customer enters some essential information in the required field such as destination, dates, number of adults/kids, etc and submit the information to the the travel agency service.

(3) The travel agency service inquires airlines, hotels or both depends on the customer preference about deals and presents them to the customer.

(4) The travel agency service presents the obtained results of the queries to the customer letting him/her to choose the best available deal.

(5) If the customer accept the offer , the the travel agency service interact with the payment services , and builds a list of options for the customer(i.e. pay full amount, buy now pay later, pay by instalment, etc)

As we emphasised above, our objective is not to present yet another WSDL/WSBPEL or even a UML description for this example, but instead we aim at discussing the importance of dynamic adaptivity and evolution in this application and the best way to tackle them. More precisely, in order to justify the main milestones of the strived proposal, with the support of this example this discussion focusses on the clarification of the following insights:

- What are business rules and how do they cope with adaptivity and evolution in Web services?
- What are the differences between adaptivity and evolution in Web services?
- Why is it crucial to distinguish between design-time and run-time changes and to have both in Web services specifications?
- What best available conceptual ingredients exist in software-engineering to tackle dynamics of adaptivity and evolution and how to suitably combine these conceptual ingredients?

Clearly this case study is one of the most requested and at the same time the most *volatile*. In addition to that this case study contain all the ingredient needed to demonstrate my work such as imported messages, exported messages, business rules and service properties.

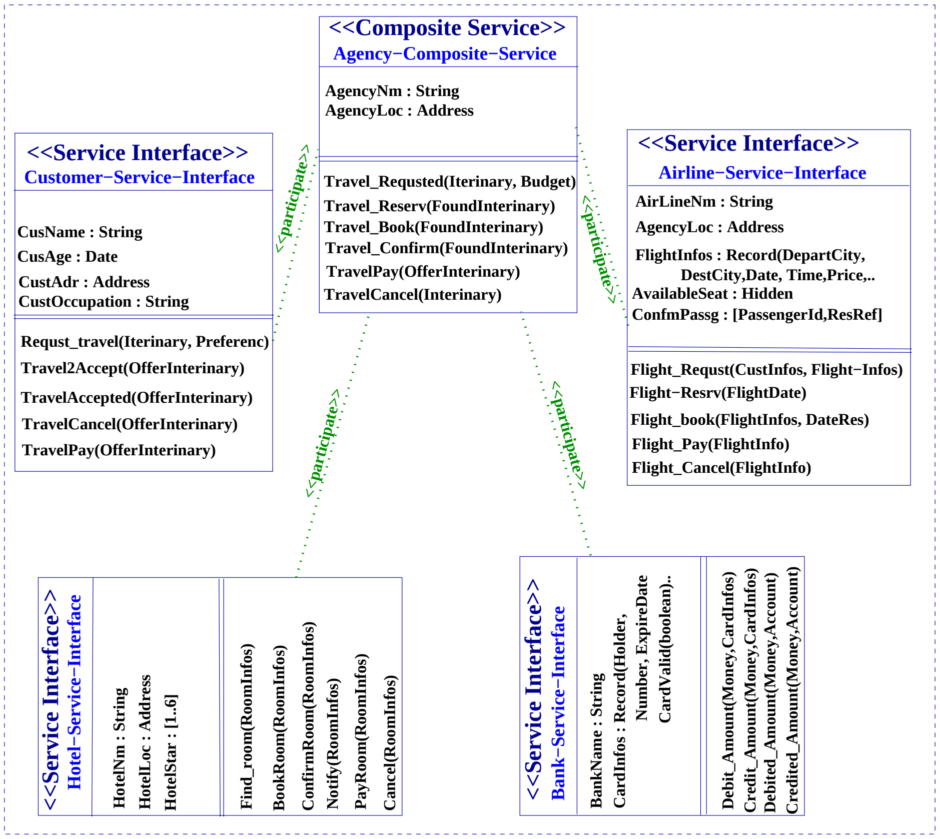

<<Composite Service>>
Agency−Composite−Service
AgencyNm : String
AgencyLoc : Address
Travel_Requsted(Iterinary, Budget)
Travel_Reserv(FoundInterinary)
Travel_Book(FoundInterinary)
Travel_Confirm(FoundInterinary)
TravelPay(OfferInterinary)
TravelCancel(Interinary)
<<Service Interface>>
Customer−Service−Interface
CusName : String
CusAge : Date
CustAdr : Address
CustOccupation : String
Requst_travel(Iterinary, Preferenc)
Travel2Accept(OfferInterinary)
TravelAccepted(OfferInterinary)
TravelCancel(OfferInterinary)
TravelPay(OfferInterinary)
<<Service Interface>>
Airline−Service−Interface
AirLineNm : String
AgencyLoc : Address
FlightInfos : Record(DepartCity, DestCity,Date, Time,Price,..
AvailableSeat : Hidden
ConfmPassg : [PassengerId,ResRef]
Flight_Requst(CustInfos, Flight−Infos)
Flight−Resrv(FlightDate)
Flight_book(FlightInfos, DateRes)
Flight_Pay(FlightInfo)
Flight_Cancel(FlightInfo)
<<participate>>
<<participate>>
<<participate>>
<<participate>>
<<Service Interface>>
Hotel−Service−Interface
HotelNm : String
HotelLoc : Address
HotelStar : [1..6]
Find_room(RoomInfos)
BookRoom(RoomInfos)
ConfirmRoom(RoomInfos)
Notify(RoomInfos)
PayRoom(RoomInfos)
Cancel(RoomInfos)
<<Service Interface>>
Bank−Service−Interface
BankName : String
CardInfos : Record(Holder, Number, ExpireDate CardValid(boolean)..
Debit_Amount(Money,CardInfos)
Credit_Amount(Money,CardInfos)
Debited_Amount(Money,Account)
Credited_Amount(Money,Account)

Figure 4.3: The Travel Agency with a SteroTyped UML Class-diagram for Services.

To specify such service-driven business applications, as with any complex reactive distributed system we have to cope with structural as well as behavioural requirements. With the de facto standardisation of UML diagrams for structural aspects (i.e. class- and object-diagrams), we argue that UML class-diagrams with slight profiled extensions allow capturing for each service, the operations and properties descriptions. The challenging problem remains the modelling of behavioural aspects, where *reactivity*, *distribution*, *compositionality* and more especially *evolution* and *adaptivity* has to be the heart of any accepted conceptual model.

At the early requirement stages, *business rules* represent the best available modelling ingredients in organisstions to cope with competitiveness and evolution. Business rules reflect regulations and conditions for the functioning of any (inter-)organisation internally as well as externally, and thus as regulations change/evolve the rules change [48, 49]. Business rules are mostly expressed in terms of Event-Conditions-Actions (ECA) forms.

The travel agency functioning has to be governed by business rules, and each service composing this application (i.e. flight service, hotel service, car rental, attraction service, etc.).

BRs for the composite vacation service: Examples of such composite business rules we may cite:

Rc1:`A vacation for X persons in a family to location Y costs C1 in case ... The vacation include ... Reservation must two weeks before`

Rc2:`The refund system is as follows ....`

BRs for the flight service: : Examples of business rules regulating flight services we include:

`Rf1: The fare we propose for a return ticket to location C is P1 for adult. For a family there a reduction for each child under ... Booking before ... cost just ...`

BRs for the hotel service: Business rules regulating flight services include:

`Rh1: Single rooms costs X1, and double rooms cost X2 for the period between ... For those staying more than D days, there is a reduction of ...`

With respect to business rules change, we distinguish between adaptivity and evolution. Both are *effects* that are *caused* by changes in the environment. For adaptivity the changes are made at run time and are seamless whilst in evolution changes happen over a long period of time and statically. For instance, we may assume refund system to change according to changes in the environment in this case delay in flight, sudden bankruptcy or change in the stock market . In contrast to that, evolution is more concerned with the introduction of new rules, the establishment of new service operations or even new complete services with their regulating function. For instance, the possibility of changing the iterinary as new rule for specific offers. We can also imagine that the travel agency proposes a new service like visiting historic sites and attractions,etc.
Besides adaptivity and evolution, distribution remains one of the essential features of service-orientation computing. It includes in our case the possibility of requesting vacation services from anywhere as well as the possibility of serving simultaneously several requests (i.e. concurrency). Related to distribution is the reactivity feature. Reactivity implies stateful modelling, where a given transaction could be long-running (especially when we require history). For instance, a customer should have the opportunity to change at any time some of the information and requirements. To cope with that, service states instances have to be explicitly represented in the model. Last but not least, while it is easy to conceive at *design-time* new services, it is more beneficial to adapt existing rules at runtime without stopping the system or decreasing its degree of distribution.
For all these considerations (i.e. distribution, reactivity, runtime adaptivity and design-time evolution), the next section presents the approach we are working

on that is based on a form of high-level Petri nets. This formal conceptual model will be automatically derived from UML-class diagrams and business rules.

4.3 CSrv-Nets: Structural Aspects Modelling of Service Interfaces

4.3.1 Service States and Messages Structure

The first step towards formalising service-driven applications consists of precisely defining different states and messages accepted by basic service interfaces as well as composite services. In the approach we are proposing, as we already emphasised we are going to benefit from advanced structuring mechanisms of the object-orientation (i.e. classification, aggregation). Also facilitate the derivation of formal service interface structures from UML class-diagrams and business rules we described in the previous section. So, in our approach besides the description of messages structure (as most of XML-technology languages offer), the precise description of service *states* enables us afterwards to specify the *statefull* concurrent behaviour of service interfaces: a capability completely missing in XML-based languages (e.g. WDSL, BPEL, etc.) and only partially addressed in recent formalisms to service specification (e.g. Petri Nets [59] Graph-Transformation [37], Temporal Logic [90], Process Algebras [22].

More precisely, we propose to specify service states as algebraic terms in the form of specific *tuples*. These service states as tuples although inspired by the structure of the OBJ language [66] object states, they enjoin very specific properties reflecting at the most the main characteristics of service interfaces. More precisely, the structure of service states we are following can be informally explained as follows:

- Any service state is conceived as a tuple composed off

$$\langle SvId \mid sv\_pr_1 : vl_1, .., sv\_pr_p : vl_p, sv_{h_1}(SvId), .., sv_{h_q}(SvId)\rangle$$

  where

  – `SvId` is interpreted as an observed service state identity where its values correspond to an appropriate abstract data type ADT (that we assume denoted as `STId`);

  – $\texttt{sv\_pr}_1, \ldots, \texttt{sv\_pr}_k$ are the observed identifiers for service state properties, which we assume having at a given time as current values respectively $\texttt{vl}_1, \ldots, \texttt{vl}_m$. We assume both service states identifiers and values to be algebraically defined (elsewhere), by denoting their respective ADT as `SPId` and `SP_Value` (as abbreviation for *S*ervice *P*roperties *Id*entifiers and *S*ervice *P*roperties *Value*s).

  – To enhance privacy, we allow the *hiding* of values of specific service state properties when required. To declare such hidden service state properties we adopt the notation of "properties-as-functions"; so if for instance the value of an property identifier, denoted by $\texttt{sv}_{h_1}$, is to be hidden, we denote it as a function $\texttt{sv}_{h_1}\texttt{(SvId)}$, with `SvId` the corresponding service state identity.

- To promote concurrency within a same service state (i.e. simultaneously performing more than one operation on different service state properties), we propose a deduction rule that *splits/ recombines* the service state at need, so that we can select at any time just the properties which are involved in the operation. This rule informally explained as follows:

$$\langle SvId \mid Sv\_prs_1, Sv\_prs_2\rangle = \langle SvId \mid Sv\_prs_1\rangle \_\_ \langle SvId \mid Sv\_prs_2\rangle$$

Sv_prs_i is an abbreviation of "$sv\_pr_i : vl_i, ..., sv\_pr_m : vl_m, sv_{h_n}(SvId), ..., sv_{h_m}(SvId)$"

- Messages involved in a given service interface are also specified as algebraic operations. Since messages act on service state instances, they should include as parameters at least one state identifier. Moreover, in a given service interface some messages may be declared to act only on states within this interface; other messages may be exported to participate in a composite service interaction (as a choreography) or take part in another service interface description (as an orchestration), and finally messages may be imported from other interfaces to constrain the messages flow in such service interface (as allowed by BPEL orchestration). In other words, in a given service interface three categories of messages may be distinguished.

Local messages: These are messages that are declared and exclusively exchanged within a given service interface. They either act for state changes in such service interface and/or allow participating and controlling the flow (i.e. the business process) of such service interface.

Imported: These messages are declared in other service interfaces and used by the given service interface in the message flow for orchestration purpose (as BPEL proposes for instance).

Exported: These messages are declared within a given service interface and used by other service interfaces or by composed services.

To bring more understandability and expressivity in our service interface formal structure, we thus explicitly distinguish between these three types of messages. This allows us afterwards to address their corresponding specific behaviour adequately. To formally capture this intuitive description of CSrv-Nets service interface structure , first we define the notion of (CSrv-Nets-)service state.

Definition 4.3.1 (Service-state structure) A service state is defined as a pair $(Sv_D \cup ST_{Sv}, \{Op\}_{ST_{Sv}})$ with:

- Sv_D is a set of (service data) sorts with at least: $\{STId, SPId, SP\_Value\} \subset Sv_D$. To allow aggregate service states, that is, service states with some properties being themselves service state identities, we define $STId$ as subsort of SP_Value (i.e. $STId < SP\_Value$).
- ST_{Sv} is a set of service state sorts (different from Sv_D), which we assume contains at least one service state sort (so we can speak about statefull service interface).
- $\{Op\}_{ST_{Sv}}$ is a set of service state operations indexed by $STId \times (SPId \times SP\_Value)^+ \times ST_{Sv}$. With each service state sort from ST_{Sv} a service state operation is associated reflecting the corresponding tuple of such service state sort.

► ***Remark 4.3.2*** As we emphasised, for the sake of understandability each service state operation indexed by $STId \times (SPId \times SP\_Value)^+ \times ST_{Sv}$ is represented as a service state tuple of the form:

$$\langle SvId \mid sv\_pr_1 : vl_1, ..., sv\_pr_p : vl_p, sv_{h_1}(SvId), ..., sv_{h_q}(SvId)\rangle$$

Where $SvId \in STId$ and $\{sv\_pr_1, .., sv\_pr_p, sv_{h_1}, .., sv_{h_q}\} \subset SPId$ and

$\{vl_1, .., vl_p, sv_{h_1}(SvId), .., sv_{h_q}(SvId)\} \subset SP\_Value$

The following presents a precise description of this service state as tuple in terms of notations inspired by the algebraic OBJ language [31].

```
    State Service-state is
      importing SP_Value  STId  SPId .
      subsort Svr_state < ST_Sv .
      subsort STId < SP_Value .
      subsort SP_Value < St_Property .
      subsort St_Property < St_Properties .
      subsort Obsv_part  Hidn_part  < Svr_state .
      op _:_ :  SPId SP_Value → St_Property .
/* observed state properties */
      op _(_) :  SPId :  STId → St_Property .
/* hidden state properties */
      op _,_ :  St_Property  St_Properties →
St_Properties [associ. commu. Id:nil].
      op ⟨_ | _⟩ :  STId  St_Properties → Svr_state .
    EndState.
```

In this description, the operator _, _ is defined in a recurrent way using the subsort property. `Svr_state` is regarded as a specific instance of the service state sorts set ST_{Sv}. As we described, service state properties may be observed or hidden . We can gather all observed (resp. hidden) properties together in new sort we call `Obsv_part` (resp. `Hidn_part`).

This important concept of service state structure leads to the concept of CSrv-Nets structure specification by extending it with different categories of service message sorts and service operations.

Definition 4.3.3 (Service-structure) The structure specification of a given service is defined as a pair $(Sv_D \cup ST_{Sv} \cup Msg_{Sv}, \{Op\}_{ST_{Sv}} \cup \{Op\}_{Msg_{Sv}})$ with:

- $(Sv_D \cup ST_{Sv}, \{Op\}_{ST_{Sv}})$ is a service state structure as defined above.
- Msg_{Sv} is a set of 'message generator' sorts different from $Sv_D \cup ST_{Sv}$. We assume that Msg_{Sv} is composed of three sets of message sorts: $\{Mes_{lo_1}, ..., Mes_{lo_l}\}$ for local message sorts, $\{Mes_{im_1}, ..., Mes_{im_m}\}$ for imported ones and $\{Mes_{ex_1}, .., Mes_{ex_m}\}$ for exported ones.
- The message operations, $\{Op\}_{Msg_{Sv}}$ is a set of message operations, that is, operations indexed by $STId^+ \times Sv_D^* \times Msg_{Sv}$. Thus, with each message sort Mes_{ij} from Msg_{Sv} a message operation (denoted ms_{ij}) is associated.

► ***Remark 4.3.4*** For syntactically describing any CSrv-Nets-structure specification we also adopt an OBJ-like language [31] as a well-known expressive (order-sorted) algebraic language. The translation into the OBJ language of the general form of such specification reflected by the above definition could be formulated as follows:

Service `Service-Structure` **is**

extending `Service-state`

subsort $\texttt{Spr}_1 \ldots \texttt{Spr}_n$ $\texttt{Sprh}_1 \ldots \texttt{Sprh}_m$ < `SP_Value` .

subsort $\texttt{Sarg}_{11,1}$ .. $\texttt{Sarg}_{l1,l1}$ .. $\texttt{Sarg}_{i1,1}$ .. $\texttt{Sarg}_{i1,i1}$ < $\texttt{Sv}_D$

subsort $\texttt{Mes}_{l1}$, $\texttt{Mes}_{l2}$,...,$\texttt{Mes}_{ll}$ < `Local_Messages` .

subsort $\texttt{Mes}_{e1}$, $\texttt{Mes}_{e2}$,...,$\texttt{Mes}_{ee}$ < `Exported_Messages` .

subsort $\texttt{Mes}_{i1}$, $\texttt{Mes}_{i2}$,...,$\texttt{Mes}_{ii}$ < `Imported_Messages` .

(* observed properties *)

op $\langle \_ \mid sv\_pr_1 : \_, \ldots, sv\_pr_1 : \_\rangle$: `STId` $\texttt{Spr}_1 \ldots \texttt{Spr}_k \rightarrow$ `Obsv_part` .

(* hidden properties or as functions *)

op $\langle \_ \mid sv_{h_1}(STId), ..., sv_{h_l}(STId) : \_\rangle$: `STId` $\texttt{Sprh}_1 \ldots \texttt{Sprh}_l \rightarrow$ `Hidn_part` .

(* local messages *)

  op $\texttt{ms}_{l1}$:STId ...$\texttt{Sarg}_{l1,1}$ ...$\texttt{Sarg}_{l1,l1}$ $\rightarrow$ $\texttt{Mes}_{ip}$.

   ... ...

(* import messages *)

  op $\texttt{ms}_{i1}$:STId ...STId ...$\texttt{Sarg}_{i1,1}$ ...$\texttt{Sarg}_{i1,i1}$ $\rightarrow$ $\texttt{Mes}_{ip}$.

   ... ...

(* export messages *)

  op $\texttt{ms}_{e1}$: STId ...STId ...$\texttt{Sarg}_{e1,1}$ ...$\texttt{Sarg}_{e1,e1}$ $\rightarrow$ $\texttt{Mes}_{e1}$.

   ... ...

EndService.

In this service description, with the notation $\texttt{Spr}_i$ we refer to the specific *sort* of the *i-th*-state property. Similarly, we denote by $\texttt{Sarg}_i$) the sort associated with the *i-th*-argument of a given service message.

4.3.2 Application to the Travel Agency

Following this CSrv-Nets- service structure, the corresponding service structure to the flight service interface, for instance, can be straightforwardly derived from the semi-formal UML service-diagram description we presented in section 2. More precisely this description is as below, where first we have to specify all imported abstract data types allowing to specify different properties and parameters of service states and their messages. These data-types should include, for instance, the city of departure and of destination (we abbreviate by `Dest` and `Depart` both sorts are string). Reservation and confirmation codes have to be specified (abbreviated by `RsvRef` and `CfrmRef`). Date of departure and of return, the maximal cost, as well as the flight fare have to be declared. Information about any customers (package), such as names, addresses, ages, number of adults, child and infants we gathered in one sort denoted $CUST\_INFO$.

Such information and more similar are crucial for expressing different business rules later in the service net behaviour. Additionally, to keep track of different passenger reservations and bookings we have previewed a list composed of the customer ID and an identifier indicating whether its reservation or booking (e.g. "Rsv" for reservation and "Bk" for booking). All this data level specification for the service flight could be declared as follows:

```
obj Airline-Data is
  protecting nat  string  date  money Time CustId
  subsort RQFLG_INFO  RSFLG_INFO < FLG_INFO
  subsort CUST_INFO < CUST_INFOS
  sort StateRSV
  subsort PSSG_RSV PSSG_CMFR PSSG < PSSGS
  subsort Dest Depart RsvRef CmfrRef  FlgRef CmfrRef < string .
  subsort DtDepart DtReturn < date .
  subsort Nb_Adult  Nb_Child Nb_Inf < nat .
  subsort Cost_Max  Fare < money .
   op [_._._._._._] :  FlgRef   Depart  Dest  DtDepart  DtReturn Cost_Max →
RQFLG_INFO
   op [_._._._._] :  RsvRef Depart Dest  DtDepart  DtReturn Fare → RSFLG_INFO
   op [_._._._._] :  CustNames CustAdrs CustAges Nb_Adult  Nb_Child Nb_Inf →
CUST_INFO
   op < _, _ > :  CustId RsvRef → PSSG_RSV
   op < _, _ > :  CustId CmfrRef → PSSG_CMFR
   op [_._] :  PSSG  PSSGS  → PSSGS

  (* These variables will be used in the behavioural part of the service net specification *)
  vars Fg :  FlghtId  ; Cs:CustId  ; Gc:AGCYId .
       Ag :  nat  ; Fr, To :  String  ; Dt:Date  ; Tm:Time
       Rs, Fm:  PSSGS  ; Mx, Cx, Py, Pn :  Money
       CsInf :  CUST_INFO  ; RqFlg :  RQFLG_INFO ; RsFlg :  RSFLG_INFO
```

endo.

Using this flight data level specification as well as the general service state description and being bounded to the general form of service structure, the corresponding service structure of the flight interface could be presented as follows. That is, first for referring service state flight sort we introduce a state sort we are denoting by `Flight_St`. Instances of flights are identified by the sort `FlghtId`. Secondly, for each message declared in the corresponding UML class-diagram, we associate a corresponding sort and an operation. For instance, for the message `FlightRequest` we declared the sort `FLGHT_RQ` and an (imported) operation we abbreviate by `FlgRq`, and which should have parameters like information about the customer, the agency ID, and clearly all detail about his/her flight iteranry and preferences. The same reasoning is to be applied to all other messages. The service state is constructed by gathering in tuple-like all properties of the flight from the UML class-diagram specification. The flight state is identified by the FlightId and is composed of the Airline name, all information about the flight (e.g. departure city, destination city, DepDate, DepTime, ArrDate, ArrTime),and the number of available seats (Denoted by `AvSeat(FlghtId)`[1]), the list of customers (IDs) reserved or booked.

```
Service Flight-Service is
  extending Service-state
  protecting AirLine-Data.
  subsort FlghtId  AirLId< STId .
  subsort Flght_St  < Srv_State
  subsort CHK_SEAT  < local_Msg.
  subsort FLGHT_RQ  FLGHT_RSV  FLGHT_BK  FLGHT_CL < imported_Msg.
```

[1]As a hidden property just to be checked.

subsort FLGHT_RQSTD FLGHT_BKD FLGHT_CLD PAY_FLGHT PAY_PNLY < exported_Msg.

(* AirLine State Properties *)

op $\langle \_ \mid AirLId : \_, FlInf : \_, AvSt(FlighId), RsvP : \_, CmfP : \_, DlRs : \_\rangle$:

FlghtId string FLG_INFOS nat PSSGS PSSGS Date→ AirLine_State.

/* Local messages */

op ChkSt : FlgId Bool → CHK_SEAT .

/* Imported i.e. received messages */

op FlgRq : CustId CUST_INFO AGCYId AirLId RQFLG_INFO → FLGHT_RQ .

op FlgRs : CustId CUST_INFO AGCYId AirLId RQFLG_INFO → FLGHT_RSV .

op FlgBk : CustId RsvRef CUST_INFO BK_INFO → FLGHT_BK .

op FlgCl : CustId AGCYId ClRef → FLGHT_CL .

/* Exported i.e. invoked messages */

op FlgRqsd : CustId FLG_INFO AGCYId AirLId RsvRef → FLGHT_RQSTD .

op FlgBkd : CustId AGCYId AirLId BkRef → FLGHT_BKD .

op FlgCld : CustId AGCYId AirLId BkRef → FLGHT_CLD .

op Payflg : CustId AGCYId BkRef money → PAY_FLGHT .

op PayPnlt : CustId AGCYId BkRef money → PAY_PNLTY .

(These variables will be used in the behavioural part of the service net specification *)*

vars Dy : Date ; Gc:AGCYId .

endo.

4.4 CSrv-Nets: Behavioural Modelling of Services

In the previous section we presented how a given service structure description denoted by $TSrv$ captures the structural aspects of a given service interface. In this section we address the behavioural concerns by incrementally constructing it from the structure description and the business rules. We refer to such behaviour issues as CSrv-Net as the behaviour is a form of high-level Petri nets tight to this service modelling and service structure descriptions. A service specification as a whole is hence a pair composed of $Serv_S P = \prec TSrv$,CSrv-Net$\succ$.

4.4.1 CSrv-Nets: Service Net Structural Features

Informally speaking, the net to be associated with a given service structure description is constructed as follows.

- The places of the net are precisely defined by associating with each service message generator one 'message' place.
- With each service state sort a 'state' place is associated.
- Transitions, which may include conditions, reflect the effect of messages on service states to which they are addressed.

► ***Remark 4.4.1*** To graphically, explicitly distinguish between local messages and external ones we draw the later with bold lines. To result in more compactness and enhancing understandability, when the effect of given messages with service states in a *given transition* results in more than one *output*—denoted by

$CsvT_1, .., CsvT_p$ (Created Service Tokens) we use boxes within such a transition to differentiate different outputs and put inside the associated condition. For instance, in most cases as depicted in flight net we will have two boxes within a given transition: one for capturing the effect of applying conditions and the second box for reporting errors and exception (i.e. the `Else` case).

Before we present the formal definition of CSrv-Nets reflecting this intuitive construction, some preliminary notations are required.

► ***Notation 4.4.2*** (1) We denote by $T_{Spr_i}(X_{Spr_i})$ the set of algebraic terms associated with each sort Spr_i (i.e. service state property, and where X_{Spr_i} is assumed to be a set of variables declared for that sort (i.e. X_{Spr_i} is a set of Spr_i-indexed variables).

(2) Given such service state property algebraic terms, we can now build algebraic terms for the whole service state. We denote such service state terms as $T_{ST_{Sv}}(X_{St})$, where X_{St} is regarded as the union of all above indexed sets of variables.

(3) In the same we denote by $T_{Msg_{Sv}}(X_{Sarg})$ the algebraic terms associated with the state message sorts, with X_{Sarg} as a union set of variables for different message argument sorts.

(4) For the sake of simplicity, we denote by X the union of all these (indexed-state properties and -message arguments) sets of variables when the distinction is not specifically required.

(5) $T_{ST_{Sv}}(\emptyset)$ (resp. $T_{Msg_{Sv}}(\emptyset)$) denotes the ground algebraic terms (i.e. without variables) for the service states and messages (as service state and

message instances). In this sense, $T_{sv}(\emptyset)$ for instance represents all service state instances belonging to the sort sv with $sv \in ST_{Sv}$.

(6) In order to capture different multi-terms (as arc-inscriptions or tokens in service places), we denote by $MT_{ST_{Sv}}(X)$ (resp. $MT_{Msg_{Sv}}(X)$) the multiset (i.e. set with the possibility of element repetitions) of terms over $T_{ST_{Sv}}(X)$ (resp. $T_{Msg_{Sv}}(X)$), with $\_\_$ as the union (associative and commutative) operation, and $\emptyset_M$ as the identity element. The two multiset forms will be subsequently referenced by $[T_{ST_{Sv}}(X)]_{\_\_}$ and $[T_{Msg_{St}}(X)]_{\_\_}$.[2]

(7) Finally to capture the whole marking (as a composition of different tokens multi-terms) of a given service specification we require another multi-terms over the above multi-terms. We denote such multi-terms as $BT_{Sv}(X)$ (resp. $BT_{Msg}(X)$) which are multisets over $SvPl \times [T_{ST_{Sv}}(X)]_{\_\_}$ (resp. $P \times [T_{Msg_{Sv}}(X)]_{\_\_}$). We assume such composite multi-terms as generated by a multi-set union operator we denote by $\_\_$ (with $\emptyset_B$ as identity). As will be precisely defined, $SvPl$ denotes a set of places in a given service behaviour specification. Elements of this multi-terms form will be referenced by $[SvPl \times ([T_{ST_{Sv}}(X)]_{\_\_} \cup [T_{Msg_{Sv}}(X)]_{\_\_})]_{\_\_}$.

Definition 4.4.3 (CSrv-Nets specification) Given a structure specification as previously defined, a Csrv-Net(-structure specification) is a structure $(SvPl, SvTr, Inp\_SvT, Out\_SvT, s, SvTC)$ where:

- $SvPl$ is a set of (service) places such that $|SvPl| = |ST_{Sv}| + |Msg_{Sv}|$. That is, the service place number corresponds exactly to the cardinality of sorts in St_{Sv} plus those in Msg_{Sv}.

[2]As we detail in a later state (resp. message) tokens correspond to ground multi-terms $MT_{ST_{Sv}}(\emptyset)$ (resp. $MT_{Msg_{Sv}}(\emptyset)$).

- $s : SvPl \longrightarrow ST_{Sv} \cup Msg_{Sv}$ is a bijection associating with each place identifier in $SvPl$ a corresponding sort from $ST_{Sv} \cup Msg_{Sv}$ as we informally commented.
- $SvTr$ is a set of (service) transitions different from the place identifiers ($SvPl \cap SvTr = \emptyset$) .
- $Inp\_SvTr : SvTr \longrightarrow [SvPl \times ([T_{ST_{Sv}}(X)]_{\_\_} \cup [T_{Msg_{Sv}}(X)]_{\_\_})]_{\_\_}$. That is, $Inp\_SvTr(t)$ can be written as $\underset{i}{\_\_}(p_i, mt_i)$, where p_i are *input* service places in the transition t, and mt_i are the associated (multisets of) tokens annotating arcs from p_i to t. Additionally, each (p_i, mt_i) must fulfill the *sort coherence* condition (of the place sort with the related input inscriptions): $mt_i \in [T_{s(p_i)}]_{\_\_}$.
- $Out\_SvTr : SvTr \longrightarrow [SvPl \times ([T_{ST_{Sv}}(X)]_{\_\_} \cup [T_{Msg_{St}}(X)]_{\_\_})]_{\_\_}$. That is, Out_SvTr captures the *output* tokens with their corresponding places, and has to fulfill the above sort coherence condition (of the place sort with the related output inscriptions).
- $SvTC : SvTr \longrightarrow (T_{ST_{Sv}}(X) \cup T_{Msg}(X))_{bool}$ is a function associating a boolean expression over $(T_{ST_{Sv}}(x(t)) \cup T_{Msg}(x(t)))$ with every transition $t \in T$; where $x(t)$ is the set of variables occurring in $Inp\_SvTr(t)$ (which as usual should include those in $Out\_SvTr(t)$).

The concept of a CSrv-Nets component as a society of instances of service states and (local/input/output) messages derived from a given structure specification, is captured by the notion of a *marked* Csrv-Net.

Definition 4.4.4 (marked Csrv-Net**)** A marked Csrv-Net is an Csrv-Net with a function $SvM : SvPl \longrightarrow [T_{St_{Sv}}(\emptyset) \cup T_{Msg}(\emptyset)]_{\_\_}$, such that if $sv \in ST_{Sv}$ then $M(s^{-1}(sv)) \in [T_{sv}(\emptyset)]_{\_\_}$. That is, each service state place contains current service states of the service specification. Otherwise, for each $Mes_{ij} \in Msg_{St}$, $M(s^{-1}(Mes_{ij})) \in [T_{Msg_{ij}}(\emptyset)]_{\_\_}$. Thus, such places contain message instances waiting for their firing by corresponding transitions.

From this notion of marked CSrv-Nets, it is now possible to define the notion of CSrv-Nets *state* which captures the distribution of markings over different (state and message) places of a given CSrv-Net.

Definition 4.4.5 (CSrv-Nets-states) Given a marked CSrv-Net as defined above, a CSrv-Net state is an element of $[SvPl \times ([T_{St_{Sv}}(\emptyset)]_{\_\_} \cup [T_{Msg}(\emptyset)]_{\_\_})]_{\_\_}$. More precisely, by denoting such state by $\mathcal{M}_{st}$, it can be written as follows: $\mathcal{M}_{st} = {}_{p_i \in \bar{S}vPl}\text{-}(p_i, M(p_i))$.

The Flight CSrv-Nets Service Structure Specification

With respect to the above flight service structure specification, the application of these behavioural constructions result in the following flight CSrv-Net behavioural interface model as depicted in Figure 4.4.
As defined above, for each service state and message sort a corresponding (typed) place is conceived. That is, to the service state sort `Flght` corresponds a service state place we denote by `Flight-St`. This service place regroups thus all flight state instances in accordance with the flight service structure specification. On the other hand, with each service message a corresponding message place is constructed. So, for the three received (i.e. imported) messages (from the agency composite service as will be detailed later) namely `Flight-Request`, `Flight-Book`, `Flight-Cancel` correspond three associated sort *messages* places. Also, for the five invoked (exported) messages places, namely `Flight-Requestd`, `Flight-Booked`, `Flight-Canceld`, `FlightPay` and

`Flight-Refund` correspond five messages. Besides that, in order to capture all different exceptions and errors related to different behaviour, we have added another message place we denote by `FlghtOP-Err`. As we will explain subsequently this place receives all attempts for violating the business rules related to different message functionalities.

4.4.2 CSrv-Nets: Service Net Behaviour Using Business Rules

The crucial contribution and added-value of our approach to the service paradigm concerns thus the concurrent behaviour that we are able to assign to different messages and services states. Such behaviour will be clearly captured by different *transitions*, with their inherent inscriptions and conditions. For that purpose, that is, for the conception of different transitions, we mainly rely on the *business rules* governing at this stage the (intra-)organisation at hand, which is the flight company in our case.

More precisely, first we propose to follow the widely dominating forms of business rules, that is, the *Event-Conditions-Actions* pattern. With respect to the formal definition of CSrv-Nets, a transition in its general pattern allows interacting some triggering messages with service states, leading to the change of invoked states, absorption of the triggering messages and apparition of new invoked messages; all this reaction is of course to be allowed under valid conditions.

For this similarity, it becomes very straightforward how a given ECA business rule can be translated into a CSrv-Nets transition that correctly reflects its behaviour. That is:

event-part: It corresponds to different input messages inscriptions involved in this event part of the rule.

condition-part: it has to be translated into a compatible transition condition. Information related to service states have to translated into input arcs inscriptions from the service state place

action-part: It is to be expressed in terms of exported messages and changes in the involved service states.

For the flight service interface, we have associated three transitions to reflect the (business) semantics of the three received messages, namely: The transition `Tfligh_rq` for capturing the request activity with the offered flights (i.e. flight_requested) as output result; the transition `Tflight_bk` for capturing the booking activity and finally the transition `Tflight_cl` to govern the cancel activity if any.

In the following, we detail the rigorous inscriptions of each of these transitions with respect to very simple business rules (BR). Afterwards, we hint how any complex business rule governing these transitions can be straightforwardly reflected into formal inscriptions in the CSrv-Nets formalism.

The Flight CSrv-Nets Service Behavioural Business Rules

A typical business rule for flight-request activity in this flight service could formulated as follows.

Rule (for flight) *"On a customer request for a flight, the offered flights have to* ***exactly*** *match the departure and destination cities, the dates and the time of the customers wish. The flight cost should not exceed what the customer tolerates and finally if the customer is a minor (less than eighteen years old) a reduction of 2% is granted".*

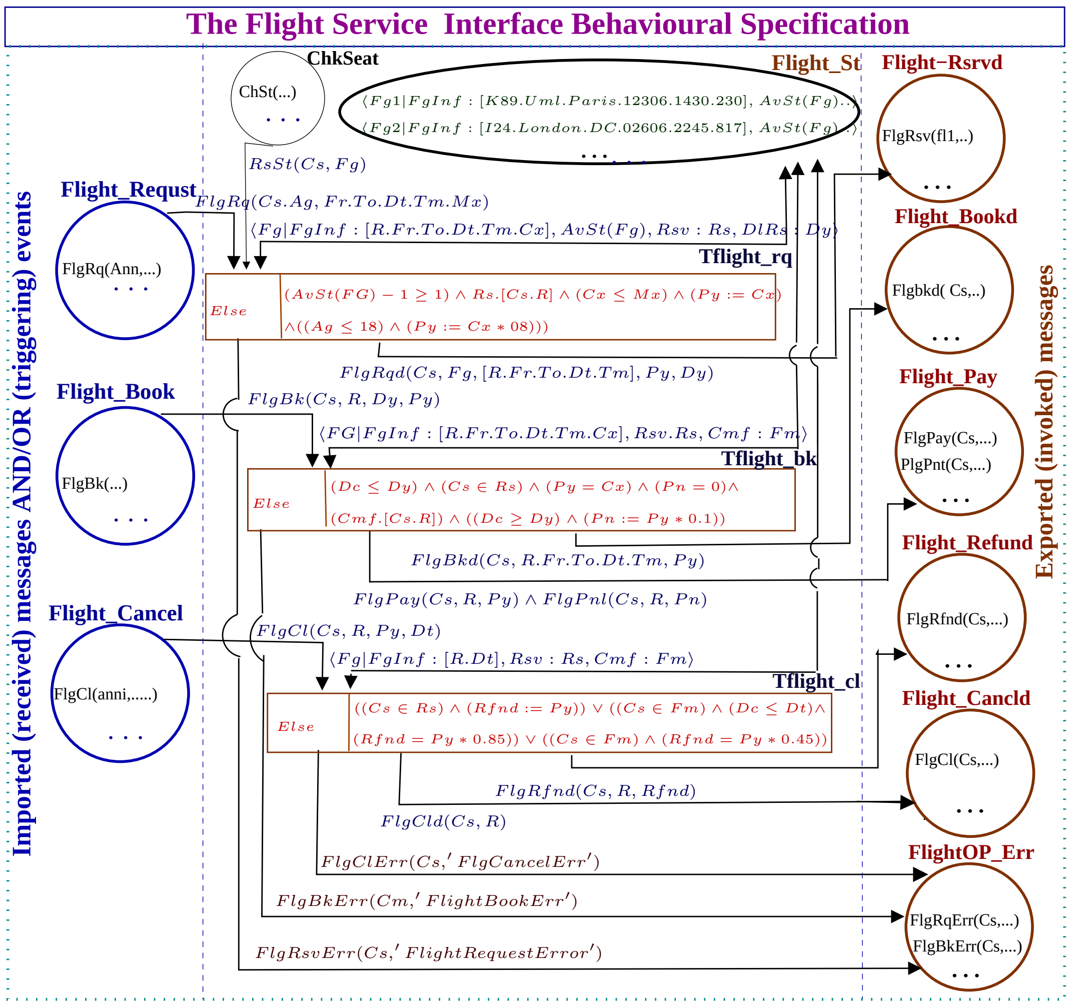
Following the above guidelines, from this informal rule description the construction of the corresponding transition (`Tflight_rq`) inscriptions could be summarised as follows:

(1) To reflect the events part of this business rule, the transition `Tflgh_rq` must have one input inscription from the request message place `Flight_Requst` and one from the state flight place `Flight_St`. By respecting the structure message description and declared variables, the inscription of the request message place takes the form: $FlgRq(Cs.Ag, Fr.To.Dt.Tm.Mx)$, that is, the parameters are "CusName, Age, From city, To city, Date, Time flight and max cost to bear. The selected (abstract) flight should have the same information (i.e. same variables). That is, the inscription from the flight state place could be: $\langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy\rangle$, with R as the flight reference, Cx as the (normal) ticket price, and Dy the date limit for booking and losing the reservation.

(2) To reflect the rule conditions, the transition condition should have the form: $AvSt(Fg) - 1 \geq 1 \wedge Rs.[Cs.R] \wedge ((Cx \leq Mx) \wedge (Py := Cx) \vee ((Ag \leq 18) \wedge (Py := Cx * 08)))$.

That is, the available seats has to be decreased by one and be still positive; The reserved list has to be updated to include the new customer and the flight reference, the ticket price Cx has to be less than customer max, and finally if the age is less than 18, the payed amount will be just 80 percent of the price.

(3) Finally, the output message to report back the founded reservation with the flight references, the computed price and the date limit. This is reflected by the inscription: $FlgRqd(Cs, Fg, R, Py, Dy)$



$\langle Fg1|FgInf : [K89.Uml.Paris.12306.1430.230], AvSt(Fg)..\rangle$
$\langle Fg2|FgInf : [I24.London.DC.02606.2245.817], AvSt(Fg)..\rangle$

$\langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy\rangle$

$(AvSt(FG) - 1 \geq 1) \wedge Rs.[Cs.R] \wedge (Cx \leq Mx) \wedge (Py := Cx)$
$\wedge((Ag \leq 18) \wedge (Py := Cx * 08)))$

$\langle FG|FgInf : [R.Fr.To.Dt.Tm.Cx], Rsv.Rs, Cmf : Fm\rangle$

$(Dc \leq Dy) \wedge (Cs \in Rs) \wedge (Py = Cx) \wedge (Pn = 0)\wedge$
$(Cmf.[Cs.R]) \wedge ((Dc \geq Dy) \wedge (Pn := Py * 0.1))$

$\langle Fg|FgInf : [R.Dt], Rsv : Rs, Cmf : Fm\rangle$

$((Cs \in Rs) \wedge (Rfnd := Py)) \vee ((Cs \in Fm) \wedge (Dc \leq Dt)\wedge$
$(Rfnd = Py * 0.85)) \vee ((Cs \in Fm) \wedge (Rfnd = Py * 0.45))$

Figure 4.4: The Behavioural Specification of flight Service Interface.

In the same spirit, we can formulate any business rules for the booking as well as the cancellation activities and construct consequently the associated net transitions (i.e. `Tflight_bk` and `Tflight_cl`). The reservation business rule should reflect the respect of the reservation deadline (ie reservation information should be still in the corresponding flight state); otherwise a penalty is to be paid beside the ticket price. The corresponding transition is easily conceived as depicted. The cancellation after a booking incur the payment of a penalty.

Following the same reasoning, the behavioural specifications of the hotel service interface, the customer service interface and the bank service interface are given in the appendix.

4.5 CSrv-Nets: Semantical Aspects

The crucial advantage of choosing High-Level Petri nets as a semantical framework for behavioural service applications is of course its ability to provide us with executable graphical animated specification, which can further by analysed. Thus the validation by rapid-prototyping at the specification level is possible against possible mistakes, misconception, inconsistency, etc.

There are a plethora of ways for interpreting the behaviour governing transitions (e.g. interleaving, process-based, operational, denotational, etc.) with respect to a given (algebraic high-level) Petri nets variant.

The operational semantics we are proposing is mainly inspired by the algebraic semantics in [79]. Informally speaking, given a transition, we have to find the right *substitutions* (i.e. the replacement of variables by corresponding complex terms for a given term t—such substitution is usually denoted by $\sigma(t)$) that show that all required tokens from the input places are available and the transition condition is evaluated to be true. In such case the marking of all involved (input/output) places is accordingly updated, that is, deletion of consumed tokens from input places and addition of newly created tokens to output places[3].

In the following we develop a more formal operational semantics for governing transition behaviour. First, to make these semantics easy and understandable, we start with an intuitive representation of any CSrv-Nets transition as a general tuple of the form:

$$\langle Transition\_Label \mid input\_inscription, output\_inscription, condition\rangle$$

With respect to our CSrv-Nets definition and for a given transition denoted t, this general tuple takes the form:

$$\langle t \mid Inp\_SvTr(t), Out\_SvTr(t), SvTC(t)\rangle$$

As we already detailed in CSrv-Nets definition, we can represent $Inp\_SvTr(t)$ as a multiset of the form: $\underset{i}{\oplus}(p_i, mt_i)$. The same for the output-inscription $Out\_SvTr(t)$ which can be captured as $\underset{i}{\oplus}(q_j, nt_j)$. Finally we denote the condition $SvTC(t)$ by its corresponding boolean multiterm.

With these details, we result in the following CSrv-Nets transition firing inference rules as the operational semantics.

Definition 4.5.1 (CSrv-Nets-transition semantics) We assume given a marked CSrv-Nets net, with its marking state denoted by $\mathcal{M}_{st} = \underset{k}{\oplus}(p_k, M(p_k))$ as defined in definition:4.4.4. Further, we assume as above that transitions are represented as a tuple:

[3]Of course test and inhibitor arcs do not involve any deletion or creation as usual

$$\langle t \mid \underset{i}{--}(p_i, mt_i), \underset{j}{--}(q_j, nt_j), [T(X)]_{bool}\rangle$$

The firing conditions and outputs are formally expressed through the following inference rule:

$$\frac{\exists\, \sigma \quad x(t) \rightarrow [T_{p_i}(\emptyset)] \mid \sigma(Inp\_SvTr(t)) = \underset{i}{--}(p_i, \sigma(mt_i)) \in \mathcal{M}_{st} \wedge (\sigma([T(\emptyset)]_{bool}) = True)}{\mathcal{M}'_{st} = \mathcal{M}_{st} - \sigma(Inp\_SvTr(t)) + \sigma(Out\_SvTr(t))}$$

With $\sigma(Out\_SvTr(t)) = \underset{j}{--}(q_j, \sigma(nt_j))$

4.6 Summary

This chapter defines *formally* a new variant of high-level Petri nets specifically designed for specifying and validating complex interacting services by focussing on their distributed and adaptive behavior. That is, instead of dealing (at design-time) with just the structural aspects in defining and composing services as currently offered by XML-Based Web technology, the variant of Service-oriented High-Level Petri Nets (we refer to as CSrv-Nets) we are proposing allows also addressing behavioral features in concurrent, adaptive and compositional way. As specific characteristics of CSrv-Nets we mainly cite the following. Firstly, CSrv-Nets is endowed with a methodology based on UML Use-Cases and class-diagrams enriched by business rules at an early stages. Secondly, CSrv-Nets allows incrementally and formally specifying and validating distributed statefull behavior of service interfaces, that we semi-automatically construct from UML-diagrams and business rules. Thirdly, interacting services are specified by composing involved service interfaces Nets, while respecting appropriate interaction patterns.Finally, the travel agency case study is followed for illustrating all CSrv-Nets aspects.

Chapter 5

Extensions of Service-based Petri Nets: Harmonious local/global Compositions

In the previous chapter, we demonstrated how service interfaces and elementary services can be progressively and rigorously leveraged to cope with concurrent rule-intensive behavioural features. We achieved that first by informally describing service structural features using stereo-typed UML class-diagrams. Then, we captured behavioural features as event-driven business rules. After constructing such informal structural-behavioural features of any service and/or interface, we then proposed for the crucial phase of rigorous formalisation an innovative service-based high-level Petri nets framework, we cleansed the CSrv-Nets conceptual model. As we incrementally introduced, CSrv-Nets allowing for precisely defining structural features with explicit service interfaces, and then smoothly capturing behavioural features.

The formalism and the whole methodology have been extensively illustrated through the Airline case-study. To recapitulate on that chapter, we may again emphasise that in contrast to Web services standards such as WSDL and BPEL, we demonstrated that instead of the very restricted static and purely process-centric service descriptions, the forwarded approach permits for formally specifying and validation services and interfaces, which are behaviourally rule-intensive, distributed and evolving.

This important achievement encouraged us to coherently push the scope of this approach one step further, so that complex *composite* services can be harmoniously tackled on top of already specified and validated individual services and interfaces using CSrv-Nets. More precisely, in terms of Web services terminology, in the previous chapter we presented how individual services (like Airlines, Hotels, etc) can be *orchestrated*in a behavioural conversational and stateful manner. Orchestration offered by Web standards such as WSBPEL means that the emphasise is put on a *single* specific service (interface) which may send/ receive messages from others. As illustrated, the behavioural specification of the Airline service using CSrv-Nets captures just the airline service, though several messages have to be exchanged with at least the customer and bank services.

In this chapter, we instead concentrate on the *choreographical* composition of several participating (elementary or complex) services to produce new complex *composite* added-value services. Web standards propose for the choreography purely static and structural descriptions through the XML-based standards such as WSCI and WS-CDL. These standards, when dealing with complex composite decentralised services are thus well-known for their limitations for coping with behavioural, adaptive and/or concurrent features characterising today's complex services. Besides that, such standards by their implementationcentricity do not provide any means for formal specification and certification, as crucial for delivering correct and reliable composite services. A typical illustration is the description of a travel-agency that requires the composition and coordination of several independent services such as: Airlines (and/or trains/taxis/car rental), Accommodation (hotel/hostel/private), financial institutions (bank/credit-card) , etc; where each of these participating are themselves very complex, conversational, stateful and adaptive.

Besides these serious deficiencies, existing proposals to Web services explicitly *differentiate* between orchestration and choreography and focus therefore strictly on either BPEL- or WSCI-related descriptions, where more investigations are being carried on BPEL-like orchestration than on choreography. This orchestration trend is mainly due to the availability of variety of advanced BPEL-driven engines, among other technological and business reasons. Indeed, choreography is surprisingly neglected with the strong belief that BPEL-like descriptions of (elementary and composite) services seem quite satisfactory in most cases. Unfortunately, with the current limited static and structural characterisations of Web standards (e.g. WSBPEL and WSCI), the striking necessity of choreography in complementing orchestrated descriptions and vice-versa have just been further confused and obscured.

The remaining sections of this chapter are structured as follows. In the next section, we bring more motivation with insights and illustration towards leveraging and complementing our approach for behavioural and adaptive service orchestration with complex choreographical compositions by extending CSrv-Nets. As first step towards conceptualising the forwarded intuitive ideas of the strived handling of choreographical composite services, the second section is devoted to the proposition of an adequate generic pattern for cross-organisational or inter-service event-driven business rules. In the third main section, we formally defined the extension of CSrv-Nets, we referred to as CCSrv-Nets, for coping with the informally developed choreographical composite behavioural and adaptive services in harmony with the orchestration of individual services. In the last section, we illustrate this sound and progressive extension of CSrv-Nets using the Agency composite service.

5.1 Choreographical Composite Services with CSrv-Nets: Further Motivation

One of the main purposes of this section consists thus in demonstrating that, when the emphasis is put on *behavioural* features rather than structural ones, such strict distinction between service orchestration and choreography and/or the focus on just one (e.g. orchestration mostly) is not only unwished but may also lead to incomplete service design and thereafter non-conformist service deployment. To cater for such essential complementation, we propose thus to optimally benefit from both service orchestration and choreography through a harmonious synergy.

Before delving into the formalisation details about how choreographically composing services as a necessary complement and sound extension to the presented behaviour-intensive orchestration, let us motivate further, from a methodological point of view, this synergy stepwise complementarity between orchestration and composite choreography, while developing knowledge-intensive service-driven applications, through the following main clarifications.

(1) By following a business rule-driven approach as we are pursuing with

CSrv-Nets, we consider that both independent elementary services as well as their composition are governed by suitable *business rules*. That is, on the one hand, we should cope with (intra-service) business rules governing (elementary/ basic) service interface behaviours, by exclusively focussing on the service at-hand. For instance, while specifying the Airline service, there are no links whatsoever to business rules about Banking, Hotel or any other services. In other words, orchestration-driven business rules are local to their respective services and interfaces. In contrast to that, (inter-services) business rules governing composite services involve different regulations, policies and strategies for correctly collaborating involved services. In this sense, the Travel-Agency composite service, for instance, has to be governed by functioning policies for optimally and legally collaborating different involved (elementary) services such as hotels, airlines, banks, etc.

(2) Having explained the existence of such two complementary categories of business rules, namely those governing intra-service behaviour and those for inter-cooperatively composing these services, the remaining central question consists in establishing the behavioural relationship between these intra- and inter-service business rules governing respectively participating services and their composition. In Figure 5.1, we projected this intrinsic relationship on the already formalised orchestration using CSrv-Nets and the to be addressed at choreographical level. On the basis of this graphical illustration, this synergic orchestration-choreography behavioural relationship, may thus be explained and motivated as follows:

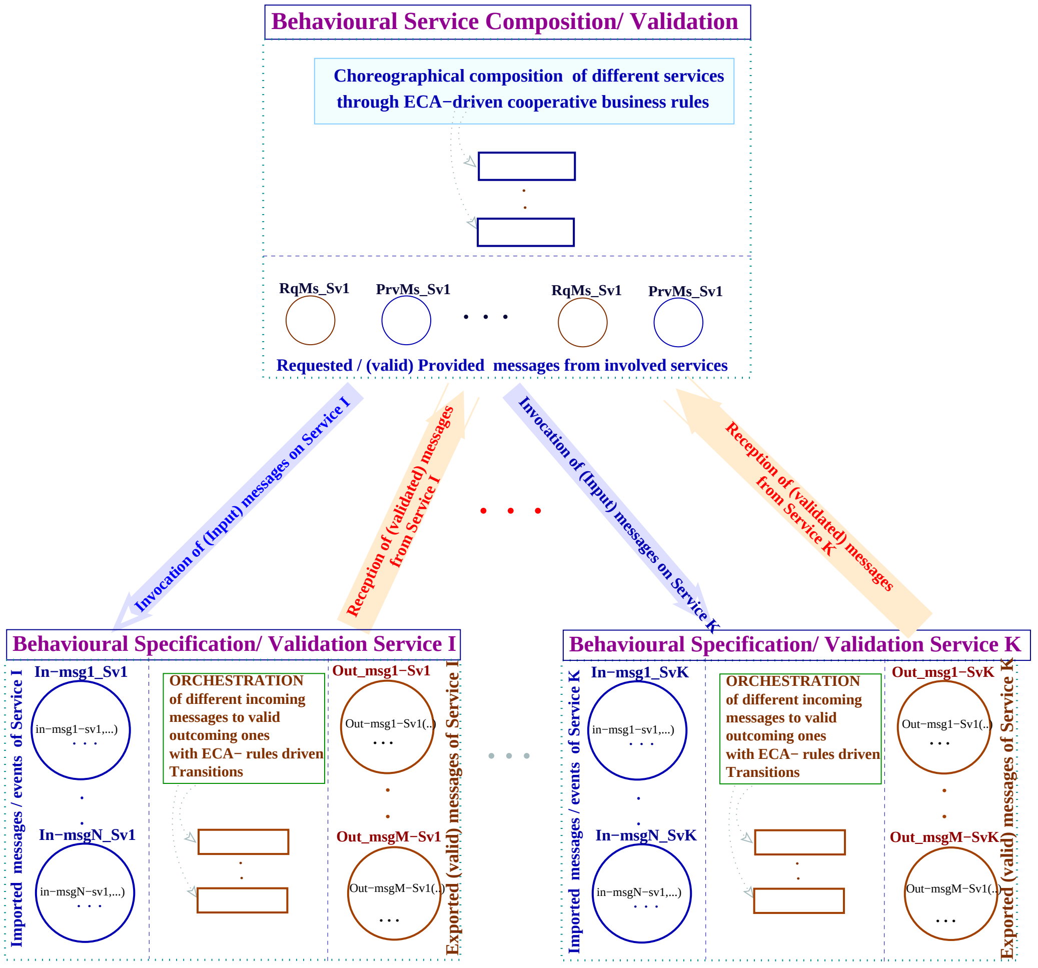

Behavioural Service Composition/ Validation
Choreographical composition of different services through ECA−driven cooperative business rules
RqMs_Sv1
PrvMs_Sv1
RqMs_Sv1
PrvMs_Sv1
Requested / (valid) Provided messages from involved services
Invocation of (Input) messages on Service I
Reception of (validated) messages from Service I
Invocation of (Input) messages on Service K
Reception of (validated) messages from Service K
Behavioural Specification/ Validation Service I
Imported messages / events of Service I
In−msg1_Sv1
in−msg1−sv1,...)
In−msgN_Sv1
in−msgN−sv1,...)
ORCHESTRATION of different incoming messages to valid outcoming ones with ECA− rules driven Transitions
Out_msg1−Sv1
Out−msg1−Sv1(..)
Out_msgM−Sv1
Out−msgM−Sv1(..)
Exported (valid) messages of Service I
Behavioural Specification/ Validation Service K
Imported messages / events of Service K
In−msg1_SvK
in−msg1−sv1,...)
In−msgN_SvK
in−msgN−sv1,...)
ORCHESTRATION of different incoming messages to valid outcoming ones with ECA− rules driven Transitions
Out_msg1−SvK
Out−msg1−Sv1(..)
Out_msgM−SvK
Out−msgM−Sv1(..)
Exported (valid) messages of Service K

Figure 5.1: An Illustrative Complementarity of Orchestration and Choreography in the CSrv-Nets Approach .

- As the Figure explicitly illustrates, we have been so far focussing on the lower part of this two-level based approach, namely the intra-service orchestration level. With respect to the running example, we have been specifying and validating the behaviour-driven (intra-)business rules of different independent services such as the airlines, the banks and hotel services. Due to the required full independency of such service interfaces from each other, at the orchestration level we could not report on how requested/provided (events and) messages in such service interfaces are to be coordinated, neither could we be able to describe the from/to where (i.e. which services) such messages are invoked or provided and for what purpose.

- With the explicit conceptualisation of this composite choreographical level over the orchestrated involved service interfaces, it becomes imperative to clarify how incoming/received and respective outgoing/provided messages (from/to different participating interfaces) are *semantically* invoked, produced and coordinated in order to fulfil the expected rule-based choreographical global behaviour in such composition. It also becomes meaningful why we have to label and identify the resulting outgoing/provided messages from such involved service interfaces as *behaviourally-certified messages*, and thereby emphasising again the necessity and benefits of such certification which as we pointed out could not be discerned with the structural limited capabilities of Web standards like WSBPEL and WSCI.

- More precisely, as depicted in Figure 5.1 by collaborating more than one service in a composite choreography, different messages have to be accordingly requested by such composite level from the participating service interfaces level. Once such messages are received by the corresponding interfaces, they have to be certified against the associated *intra-service business rules* (using corresponding CSrv-Nets-driven transitions). That is, invoked messages at different services can either result in a behaviourally-certified outgoing messages or result in non-conforming incorrect exception messages. Only behaviourally-certified (through service interfaces orchestration) messages can further be invoked at the choreographical composition level to realise the expected added-value collaborative business activities with respect to the business process at hand.

- Another benefit of this two-level rule-driven approach to service applications development is that besides usual functional rule-based requirements, at the composite choreographical level, *non-functional* rules and policies can also be straightforwardly applied and enforced on the participating services. For instance, we may impose a response time on a given activity (request for flight or accommodation, etc.) or invoke only specific airlines depending on reputation, trust, etc.

5.1.1 Choreographical Composition within the Travel-Agency

Concretely with our vacation running case study, the composite Travel-Agency once receiving a validated request from the customer (i.e. can ask for a vacation with validated conditions such as the age, etc.), depending on general agencies regulation and policy rules (besides context-aware incentive offers specific to each particular agency), the agency dispatches then different request messages to involved services to get Tickets, Accommodation, etc. In contrast to usual (WSBPEL or WSCI) Web standards, with our two-level behaviour-intensive service approach there are *no assumed systematic "positive" replies* from such invoked service interfaces, even when fully available. That is, to get Tickets, Accommodation, etc, different current local behavioural business rules put in place have to be checked and validated, and only in the positive case, the

Agency can proceed further (with the check-out of bank in the same way, etc). Towards a more conceptualisation of this rule-based choreographical behaviour in composing services, we first propose a general pattern for required cross-organisational choreographical level business rules. Then, we formally extend the CSrv-Nets framework to rigorously reflect such interaction-driven business rules, which are more close to *transient architectural connectors* [77]. Finally, we illustrate this crucial behavioural conceptualisation of choreography with our running travel agency.

5.2 Business-Rules Pattern for Behavioural Choreography

For capturing cross-organisational composite business rules, unlike intra-service business rules, we have to take into consideration besides the ECA rule itself several other clauses with the following determinant ingredients:

Participant services: In this clause, we have to precisely set different service (interfaces) types (with some of their instances when needed) taking part in the composition. Once a (behaviourally specified) service interface is stated to be part of a given composition, all its incoming/outgoing messages can be requested /provided by the composite level; they all participate in formulating any inter-service choreographical business rules. Besides that, specific properties from participating states, such as identities and other properties, may be part of the composition. In terms of architectural techniques [77], this part of the cross-organisational rules corresponds to the concept of roles in architectural connectors.

Extra proper properties for interacting: Depending on the composition of behavioural semantics, additional extra composition-driven properties such as messages, (stateful) property are to be declared for expressing the intended composition.

ECA-like effects on services: This main part formulates the rule itself, where we have first to express the events triggering such rules, then the conditions to be observed by the composition (in terms of constraints on the participating service properties and proper ones as well as messages to be exchanged between involved services). Finally, actions in terms of messages to perform on different partners and on the composition itself have to be explicitly defined following the intuitive semantics of the rule at-hand.

Towards expressing this behaviour-driven choreography in a disciplined but still intuitive way while explicitly coping with the above constituents, we propose a general pattern for such cross-organisational behavioural business rules as event-driven architectural connectors. This general pattern respects the following form.

Choreographical ECA-behaviour <*Service-Composition-Identifier*>
  participant interfaces <*list-of-service interfaces*>
  invariants <*possible extra-interaction constraints*>
  properties/messages <*possible extra ingredients for interaction*>
  interaction rule: <*Rule-Name1*>
    at-trigger <`(set-of-)events`>
    under <`cross-partner conditions`>
    acting <`set-of-actions-and-events to perform and trigger`>

As we emphasised, important in this behaviour-driven services composition is above all the name of participating service interfaces. Secondly, when required we have to specify additional invariants and constraints to be observed during the interaction. Thirdly, besides exchanged messages, stateful data and events from the participants, to express complex interaction patterns we can define at need additional properties such as constants and messages at the composition level. The ECA-based interaction rule itself starts by describing the event(s) triggering the interaction, then which conditions have to be fulfilled and finally what are the cooperative actions to be performed. Notice that in some cases, among the actions we may have triggering events to direct, initiate other semantically related composite rules. This behavioural business rules pattern requires of course from different participating entities explicit interfaces including different events, messages and other properties (such as constants, variables ,etc). Such participating service interfaces, as we already motivated, need thus to be already specified and certified (i.e. orchestrated) in order to take part in a given choreographically-driven cross-service business rules.

5.2.1 Cross-Services Business Rules in the Agency Application

To stay competitive, travel agencies are steadily offering different incentive packages for their customers. For instance, depending on the customer profiles (e..g trust, frequency, status, individual or group, etc) different attractive offers can be provided. These vacation packages represent in fact the main cross-organisational business rules regulating the cooperate behavioural functioning of travel agencies, with respect to participating services such as: Accommodation (hotels, hostels, apartments, etc), transportation (e.g. airlines, trains, car rental, etc), attractions (visiting sights, attraction places) and financing (credit cards, banking, etc), among other participants. A typical rule governing a given travel agency functioning, could be expressed as follows:

Travel-Rule1 *"For a group of persons taking a vacation, different formulas are proposed to them. If more than two persons, traveling to Location-X and booking T-weeks before their departure (with* $T \geq 2$ *for instance), they are eligible to a specific percentage reduction, we denote by P with* $10 \geq P \leq 30$ *as an illustration. When they decide for specific accommodation they get extra A reduction percent, and when they pay with credit cards, they get more C-percent. Finally, if they stay more than W weeks, they get extra K-percent".*

In order to describe this simple cross-organisational rule in compliance with the above general rule pattern, we further require more specific information from the composing travel agency service. First, besides its name and address, an agency should have usually favoured airline partners, favoured accommodation hotels (series), and not least favoured destination locations with corresponding basic prices for specific periods (for individual customers). For the sake of exhibiting more behaviours, we assume that customer requests can only be processed for such privileged partners and destinations; otherwise there will be for instance no discounts and promotion . Taking this agency knowledge into play as well as the capabilities of the three involved services (e.g. airlines, accommodations, banks and also the user service), the above agency rule could be enforced in fact through three steps and thus has to be split in some sense into three "sub-"rules.

(1) The first sub-rule concerns the `request` activity, where the agency by receiving the event `Request_Travel` from the customer service, has to check

whether the destination belongs the privileged ones, and more importantly whether the "spending-threshold" amount set by the customer is within the range of the expected total costs. In the affirmative case, the agency submit a corresponding request like `Find_Travel`. This composite request message should involve different request messages for tickets, accommodation and other involved services (e.g. attraction visits, car rental, etc.) depending on the wish of the customer.

(2) The next sub-rule has to capture the acceptance/refusal of best offers (from these participating services). That is, once receiving different offers from the involved services (e.g. airline booking, accommodation booking, etc), the agency asks the customer service to confirm or inform the composite complete offer, which also has to take into account the afore-described reductions and promotions. That is, the main core of the afore-described business rule has to be enforced at this stage, resulting in the effective booking of air-tickets, hotel-rooms, etc by respecting the local business rule at each corresponding service interface.

(3) The final phase in establishing this rule-driven vacation composite service consists in positively receiving the final booking from different requested services. In such case, the customer has to proceed to the payment. Once such payment is accepted (i.e. by credit-card or through bank-transfer, etc) the travel is enabled through the handing of tickets, etc.

(4) Finally, it is worth pointing out that the customer even after endorsing the acceptance of the proposed travel offer, can still cancel it. Nevertheless, customer must pay in this case an accordingly rule-driven penalty amount.

Choreographical ECA-behaviour `Travel_Agency`

```
participant services
    Flg:  Airlines
    Acom:  Accommodation
    Cust:  Customer
    Bank:  Bank
constants
      Prc1, Prc2:  [0..1]
properties
      PvgTRP:List[Dest,{Cost_Range.Duration}, ValidTil]
      PvgFLG:List[AirL, Dest, Fare_Range, ValidTil]
      PvgACM:List[Dest, {Star.Fare_Range},ValidTil]
messages
      Trip_ToFind Trip_Found Trip_Book Trip_Booked
      Trip_Pay Trip_Paid  Trip_Cancel Pay_Penality
interaction rule :  Found_Trip (sub-rule1)
at-trigger Cust.Trip_Reqstd(CsInf,TrInf, Mx_Cst)
   under (TrInf.Typ='TRIP')
    if (TrInf.Dest ∈PvgTRIP.Dest) and (Mx_Cst ≤PvgTRIP.Cost_Range)
      and (TrInf.DepDt ∈PvgTRIP.ValidTil)
      let (var P = [TrInf.ReturnDt-TrInfo.DepDt])
        if (PvgTRIP.{CxT.P}) and (TrInf.DepDt-DtNow≥30))
   acting Flg.Flg_Requst(CsInf, FlgInf, 3/4*Prc1*CxT) and
      Acom.Hotel_Requst(CsInf, HotlInfo, 1/4*Prc1*CxT)
      and "Trip_ToFind(AgInf, DtNow)"

   under (TrInfo.Typ='FLIGHT')
    if (TrInfo.Dest ∈PvgFLG.Dest) and (Mx_Cst ≤PvgFLG.Fare_Range)
      and (TrInf.DepDt ∈PvgFLG.ValidTil) and (TrInf.DepDt-DtNw≥14))
   acting Flg.Flg_Requst(CsInf, FlgInf, Prc2*PvgFLG.Fare_Range) and

interaction rule:  Choose_Trip (sub-rule2)
```

at-trigger Cust.Trip_Choosed(FlgRefi,HtlRefj) **and**

  {Flg_Rsvd(Cust, FgRsv_Info)} **and** {Htl_Rsvd(Cust, HtRsv_Info)}

 under (FlgRef=Min(allFgRsv_Info.Fare) **and** (HtlRef=Min(allHtRsv_Info.Price)

 acting Cust.Trip2Confirm(Flg_Rsvd(FlgRef$_i$), Htl_Rsvd(HtlRef$_j$))

interaction rule: **Confirmed_Trip (sub-rule3)**

at-trigger Cust.Trip_Confrimd(CsI, FlgRef$_i$,HtlRef$_j$)

 under (True)

 acting Cust.Trip2Book(Flg2Bk(CsI, FlgRef$_i$), Htl2Bk(CsI, HtlRef$_j$))

interaction rule : **Booked_Trip (sub-rule4)**

at-trigger Cust.Trip_Booked(CsI, FlgRef$_i$,HtlRef$_j$)

 under (True)

 acting Cust.Trip2Pay(Flg2Bk(CsI, FlgRef$_i$), Htl2Bk(CsI, HtlRef$_j$), Cost)

end Std-withdraw interaction rule: **Cancel_Trip (sub-rule5)**

at-trigger Cust.Trip_Cancel(CsI, FlgRef$_i$,HtlRef$_j$)

 under Trip_Booked(CsI, FlgRef$_i$,HtlRef$_j$) = True

 acting TripCancelled(CsI, TrRef$_i$)**and** Cust.Penalty2Pay(CsI, TrRef$_i$,
Cost-Penalty(formulas))

end Std-withdraw interaction rule: **Pay_Trip (sub-rule6)**

at-trigger Cust.Trip_Pay(CsI, TrpRef$_j$, Cost(Trip))

 under Trip_Booked(CsI, FlgRef$_i$,HtlRef$_j$) = True

 acting Bank.TripPaid(CsI, Cost(Trip))

end TravelAgent_Rules

As depicted in the top of the choreography-driven cross-service business rules, four partners are involved namely the airlines, accommodation, customers and the banking services (besides the agency itself). In order to allow explicitly manipulating (i.e. invoking/receiving) messages from these partners, we declare any required instances. In our case, we declared the service Flg for airlines, Acom for accommodation, Cust for Customer and finally Bank as an instance of the bank service.

As we emphasised, usually within each Travel agency there are seasonal offers such as a list of privileged trips with attractive prices, list of selected accommodation and/or list of privileged flights and airline partners. Each element in these lists is composed with all necessary information; so, for instance, for a given privileged trip one may find the destination, a range of costs depending of the duration and the date limit of such offers. The same is to be observed for the privileged flights and accommodation. We have regrouped all this specific information as properties at the composite service level. In addition to these variables, to capture the different discount percentages, we have defined some constants (e.g. `Prc1`, `Prc2`) those values ranging over the internal [0..1], that is real values between 0 and 1.

Besides these stateful static properties, with the aim to promote the adaptability, we conceived a set of messages that allow *factoring out* different invoked/received messages from the participating services. In this sense, for instance, through the (generic abstract) the message ”`Trip_ToFind`” we are factoring out all involved requested messages (from different changing partners) such as: `Find_flight`, *Find_accommodation*, and so on. In such a way, we can thus add any other respective message when incorporating another service such as `Find_RentingCar` or `Find_TouristSite` *without modifying* this generic message `Trip_ToFind`.

Towards modelling business rules for trips as choreographical service composition, as motivated we propose to proceed in a conversational process-centric way, where the behaviour of each involved business activity is captured through associate cross-service business rules. These business activities consist of: (1) Look for candidate trips (e.g. associated flights, accommodations, etc); (2) choose the best trip from the found candidates; (3) Confirm the selected trip ; (4) Pay for that trip or Cancel it.

The principled (architectural) description of the first business rule governing the business activity "Look for candidate trips" may be explained as follows. This business rule has to be triggered through a "successful" request from customer service, where information about this requester, the desired trip details as well as the maximal cost that can be spent on such a trip. In this context the "successful", request event from the customer, means that constraints and conditions required from any customer have been already checked at the customer service level (such as the age, address, etc using our CSrv-Nets framework). Please, note that to explicitly indicate that such message is to be received from the *customer* service, we have prefixed it with the variable `Cust`, which refers to an instance of a customer service. Once such triggering trip-request message has been recognised by the agency, the first element to check from the trip information is the type of the trip, that is: is it just a flight?, accommodation? or a complex composite trip?. Afterwards, we have to check whether the requested trip (in case of a requested trip) belongs to a privileged trip that is still valid. In that case, we have also to ensure that the proposed customer threshold budget is not surpassed. Under such minimal required constraints, we can now discuss different forms of possible discounts so that we set the maximal fares to be enforced while requesting candidate flights and accommodation (by sending messages to respective services). We have depicted one of such discount computations. That is, if the trip request is made four weeks before traveling (i.e.

30 days), then the threshold fare to be enforced while requesting flights should be $3/4 * Prc1$ percent of the normal cost of such privileged trip (i.e. CxT). The price for the requested accommodation should in consequence not be more than the $1/4 * Prc1$ of that price.

In the spirit of such modelling, one can further define any possible business rules for imposing different discount regulations depending not only on the duration and date of request, but also on the period such as during the summer,christmas and so on. For the case of just a flight instead of a complete trip (i.e. no accommodation) a similar discount formula has been suggested (i.e. when requested before two weeks then a discount of `Prc2` of the suggested fare has to be observed while requesting candidate flights).

In the way we have described the corresponding business rules for the other business activities such as: `Trip_Selection`, `Trip_Confirmation`, `Trip_Booking`, `Trip_Cancelling` and/or `Trip_Paying`. We skip their descriptions here and let them be a simple exercise for the reader.

5.3 Leveraging CSrv-Nets to Choreographical ECA-Driven Business Rules

After motivating and intuitively presenting how composing services in cooperative cross-organisations on the basis of behavioural-intensive ECA-driven business rules, and once this individual involved services interfaces have been specified and validated, the purpose of the following is to leverage these intuitive descriptions to a more rigorous level. As we pointed out, we aim at achieving that by soundly extending the so far CSrv-Nets framework so that it can explicitly specify and validate such dynamically interacting services. In the same spirit as we proceeded for the CSrv-Nets presentation, first we address the structural features in composing different CSrv-Nets interface specifications, then we formalise the behavioural features and finally illustrate this composite CSrv-Nets formalism (we refer to as CCSrv-Nets) with the same travel agency example by incrementally translating the above motivated ECA cross-organisational business rules to this new choreography-driven formalism.

5.3.1 Structural Features in CCSrv-Nets

As explained, in most cases, to achieve a conversational and behaviour-driven composition we require *proper* properties, besides requested/received messages from the participating services. These proper properties may be (observed) messages and events to be declared at the composition level. As for the specification of CSrv-Nets interfaces, we propose to adopt the same algebraic specification pattern for formally specifying them. That is, all proper properties are gathered in a tuple form, we refer to as the composition state type, while each message or event is explicitly specified as an algebraic operation.

All participating service interfaces (i.e. their algebraic specifications) have to be explicitly included in the composition specification, using for instance, either the usual primitives such as `including` or more expressively the keyword `participants`, to explicitly highlight them as services participating in this choreographical knowledge-based composition.

A further crucial enrichment with respect to the usual CSrv-Nets structural specification concerns what we may refer to as the ”gathering” or integration of several requested/provided message types into a single new (composite and flexible) message type. To motivate and then define this notion of a unified message, let us consider again the the Travel-Agency case with in mind the choreography general form we depicted in Figure 5.1. We clearly observe that typical agency activities such as ”`Request-Trip`”, ”`Trip_confrim`” and/or ”`Trip-Select`” all consist in accordingly (e.g. conforming to corresponding business rules) *dispatching* or sending requesting messages to corresponding participants (e.g. airlines, accommodation, attractions, car-rental, etc.) or collecting/receiving certified provided messages. More precisely, participating requested messages such as `request4flight`, `request4hotel`, `request4car`, etc, are intrinsically and semantically meaningful *only together* in that they all concern the ”Request4Trip” activity, in the same way unified messages apply to Select4Trip, etc. In terms of algebraic concepts, we declare such unified message types as `super-sorts` of respective detailed service-related messages. So for instance, the unified message type for `request4Trip` will be a super-sort for all message types concerning service requests such as: `request4flight`, `request4hotel`, `request4car`, etc. In this way, we are promoting by further adaptability and flexibility at the choreography composition level. This high flexibility comes from the fact that, for instance, `Request4Trip` or `Trip2Confirm` unified composite message now depend directly on the customer’s wishes. For instance, for customers requesting just flights, the composite message `Request4Trip` will include just the `request4flight`, and so on. In this way, we are indeed dynamically reshaping this composite message depending on the participants services. At this syntactical structural level, we use the symbolic notation $\ll$ instead of the usual subset symbol $<$ to distinguish this notion of a composite message. At the behavioural Petri Nets composition level, there are many fold benefits and semantics of this notion of unified messages as will be detailed later.

The Agency Structural Aspects in CCSrv-Nets

Capitalising on the discussed cross-service rules governing the composite Agency service, we conceive here the formal structure of the resulting CCSrv-Nets. We should mention that for sake of more clarity we are adopting some extra-notations such as: (1) Instead of the obj-primitive "include" we are using the keyword "participating" to emphasise the interaction-centricity of that structure (compared to our previous CSrv-Nets); (2) instead of the usual subsort < we are using the super-sort ≪; and (3) of the keyword "obj", we are using the new expressive keyword "`Composite-Service`". Moreover, we should recall that as usual all required Data as assumed to be defined elsewhere. For that purpose, we have gathered them in a data-level data type named "AGENCY-Data", we just import (in a protected mode) to that CCSrv-Nets structure specification.

```
Composite-Service Agency-Service is
  extending Service-state
  protecting AGENCY-Data.
  participants Customer, Airline, Acommodation, Bank < Service .
  supersort Trip_ToFind ≫ Flg_Requst  Room_Request .
  supersort Trip_Found ≫ Flg_Reservd  Room_Rservd .
  supersort Trip_ToBook ≫ Flg2Book  Room2Book .
  supersort Trip_Booked ≫ Flg_Bookd  Room_Bookd .
  supersort Trip_Cancel ≫ Flg2Cancel  Room2Cancel.
  subsort TRP_CHOS  TRP_CFRM  TRP_BOK  TRP_PAID  TRP_PENALTY TRIP_START
  (* Agency State Properties *)
    op ⟨_ | AgcNm : _, PrvTrip : _, PrvFLG : _, PrvACOM : _, RsvList : _, CmfList : _,
 CanclList:_⟩ :
          AgcID  string  LIST-PrvTrip  LIST-PrvFLG  LIST-PrvACOM  RsvLIST
```

```
CfmLIST CancelLIST
/* messages */
   op Trip_Choose :  AgcId CustRef FlgRef → TRIP_CHOS .
   op Trip_Confrm :  AgcId CustRef FlgRef → TRIP_CFRM .
   op Trip_Book :  AgcId CustRef FlgRef → TRIP_BOK .
   op Trip_Pay :  AgcId CustRef FlgRef Cost → TRIP_PAID .
  (* These variables will be used in the behavioural part of the service net specification *)
  vars Tpv :  PrvTRP; Fgv :  PrvFLG ; CsI : CustID .
       fg_i :  FlgRef ; ht_i :  HotRef ;
       Dt, DepDt, Now :  Date
       MxC, Cxt, CsTT : Money
Composite-Service.
```

5.3.2 Behaviourally Composing Services with CCSrv-Nets

With CSrv-Nets capabilities in capturing stateful service interfaces behaviour, we present in the following how such choreographical ECA-driven behavioural rules enhance these potentials towards more dynamic adaptivity and evolution. Following the same intuitive guidelines for constructing CSrv-Nets service interfaces behaviour from informal service applications, the modelling steps for integrating such choreographical architectural behaviour on top of involved CSrv-Nets service interfaces could be sketched in the following. First, we have to derive from a given ECA-based architectural connector description, a more precise corresponding service description specification by algebraically specifying different properties (messages, events, etc.) as we did exactly for CSrv-Nets. Secondly, by gathering different composite service properties and participants into states, we then associate it a type as we explained above and a corresponding *place*. In the same way for each (non-unified) declared message in the composite service, we associate a corresponding message place. For the so-called unified messages (i.e. those followed by the super-sort symbol $\gg$ as described above) we associate a fusion place (as defined in Coloured Petri Nets for instance [42]). A Fusion place is a place that contains more than other places. The enclosed places are those corresponding to the sub-sort message names (from other services).

Before illustrating this intuitive translation towards CCSRV-NETS, let us present a more rigorous definition of this formalism of CCSRV-NETS. For that purpose, we present how to syntactically define the concept of composite service from participating CCSRV-NETS behavioural service interfaces.

5.4 Application of the Approach to the Running Example (Travel Agency)

Following the above informal guidelines on how to construct any choreographical CCSRV-NETS as well as the detailed cross-service business rules governing the travel agency, we bring here further explanations on the resulting CCSRV-NETS travel-agency conceptual model as depicted in Figure 5.2.

First for each message we are associating a place. In this sense, for instance, for the message type `TripRqtd` we are associating the place `Trip_Reqstd`. Moreover, as we defined in the structural part some places are considered as unified (i.e. `Trips_Found`), gathering more than one message place (ie the flight and room request message places). For each business rule, a corresponding transition governing a business activity is conceived. For instance, with the business

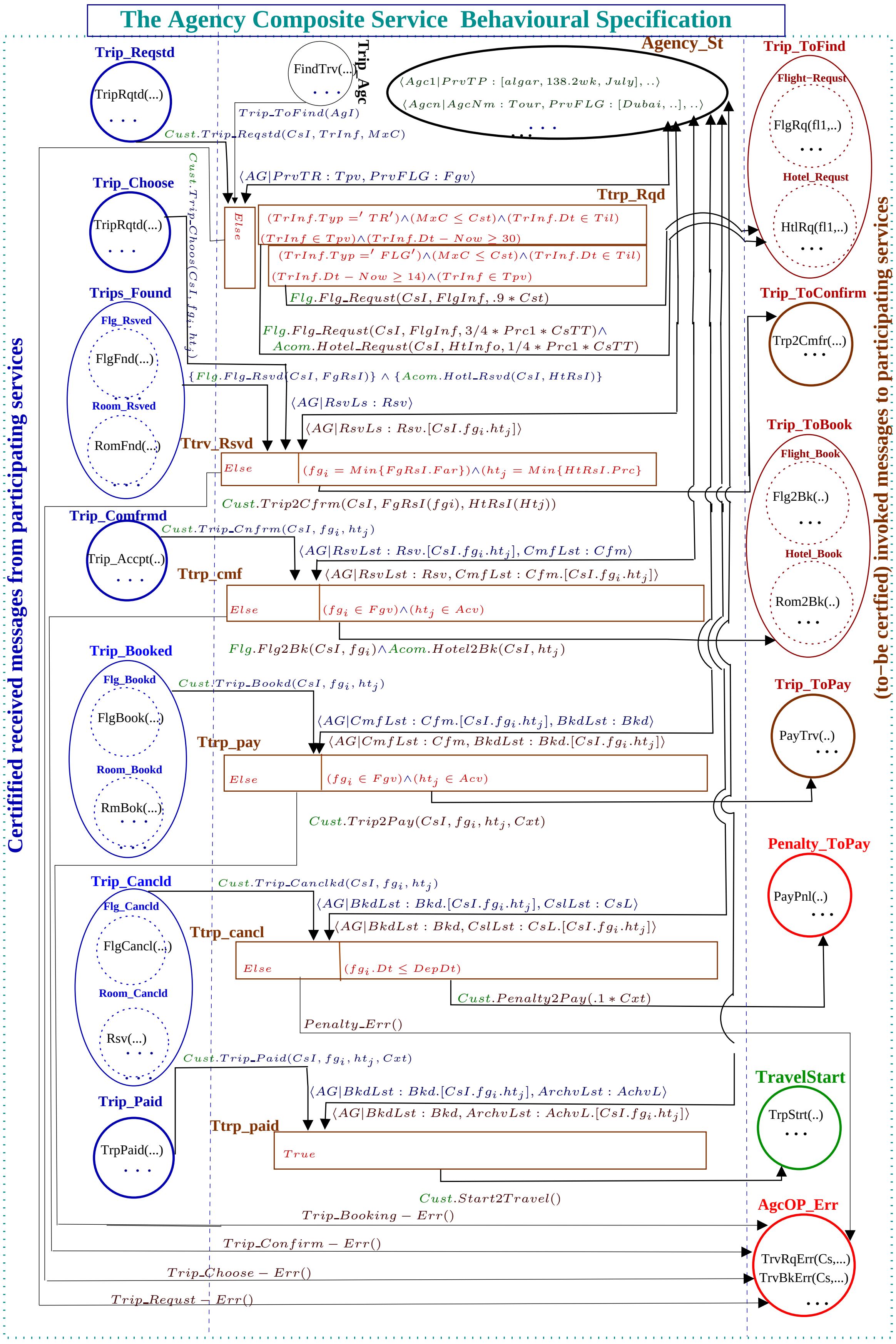

The Agency Composite Service Behavioural Specification
Certified received messages from participating services
(to-be certfied) invoked messages to participating services
Trip_Reqstd
TripRqtd(...)
FindTrv(...)
Trip_Agc
Agency_St
$\langle Agc1|PrvTP : [algar, 138.2wk, July], ..\rangle$
$\langle Agcn|AgcNm : Tour, PrvFLG : [Dubai, ..], ..\rangle$
$Trip\_ToFind(AgI)$
$Cust.Trip\_Reqstd(CsI, TrInf, MxC)$
$\langle AG|PrvTR : Tpv, PrvFLG : Fgv\rangle$
Trip_Choose
TripRqtd(...)
$Cust.Trip\_Choos(CsI, fg_i, ht_j)$
Else
Ttrp_Rqd
$(TrInf.Typ =' TR')\wedge(MxC \le Cst)\wedge(TrInf.Dt \in Til)$
$(TrInf \in Tpv)\wedge(TrInf.Dt - Now > 30)$
$(TrInf.Typ =' FLG')\wedge(MxC \le Cst)\wedge(TrInf.Dt \in Til)$
$(TrInf.Dt - Now \ge 14)\wedge(TrInf \in Tpv)$
$Flg.Flg\_Requst(CsI, FlgInf, .9 * Cst)$
$Flg.Flg\_Requst(CsI, FlgInf, 3/4 * Prc1 * CsTT)\wedge$
$Acom.Hotel\_Requst(CsI, HtInfo, 1/4 * Prc1 * CsTT)$
Trip_ToFind
Flight-Requst
FlgRq(fl1,..)
Hotel_Requst
HtlRq(fl1,..)
Trips_Found
Flg_Rsved
FlgFnd(...)
Room_Rsved
RomFnd(...)
$\{Flg.Flg\_Rsvd(CsI, FgRsI)\} \wedge \{Acom.Hotl\_Rsvd(CsI, HtRsI)\}$
$\langle AG|RsvLs : Rsv\rangle$
$\langle AG|RsvLs : Rsv.[CsI.fg_i.ht_j]\rangle$
Ttrv_Rsvd
Else
$(fg_i = Min\{FgRsI.Far\})\wedge(ht_j = Min\{HtRsI.Prc\}$
$Cust.Trip2Cfrm(CsI, FgRsI(fgi), HtRsI(Htj))$
Trip_ToConfirm
Trp2Cmfr(...)
Trip_Comfrmd
Trip_Accpt(..)
$Cust.Trip\_Cnfrm(CsI, fg_i, ht_j)$
$\langle AG|RsvLst : Rsv.[CsI.fg_i.ht_j], CmfLst : Cfm\rangle$
$\langle AG|RsvLst : Rsv, CmfLst : Cfm.[CsI.fg_i.ht_j]\rangle$
Ttrp_cmf
Else
$(fg_i \in Fgv)\wedge(ht_j \in Acv)$
$Flg.Flg2Bk(CsI, fg_i)\wedge Acom.Hotel2Bk(CsI, ht_j)$
Trip_ToBook
Flight_Book
Flg2Bk(..)
Hotel_Book
Rom2Bk(..)
Trip_Booked
Flg_Bookd
FlgBook(...)
Room_Bookd
RmBok(...)
$Cust.Trip\_Bookd(CsI, fg_i, ht_j)$
$\langle AG|CmfLst : Cfm.[CsI.fg_i.ht_j], BkdLst : Bkd\rangle$
$\langle AG|CmfLst : Cfm, BkdLst : Bkd.[CsI.fg_i.ht_j]\rangle$
Ttrp_pay
Else
$(fg_i \in Fgv)\wedge(ht_j \in Acv)$
$Cust.Trip2Pay(CsI, fg_i, ht_j, Cxt)$
Trip_ToPay
PayTrv(..)
Penalty_ToPay
PayPnl(..)
Trip_Cancld
Flg_Cancld
FlgCancl(...)
Room_Cancld
Rsv(...)
$Cust.Trip\_Canclkd(CsI, fg_i, ht_j)$
$\langle AG|BkdLst : Bkd.[CsI.fg_i.ht_j], CslLst : CsL\rangle$
$\langle AG|BkdLst : Bkd, CslLst : CsL.[CsI.fg_i.ht_j]\rangle$
Ttrp_cancl
Else
$(fg_i.Dt \le DepDt)$
$Cust.Penalty2Pay(.1 * Cxt)$
$Penalty\_Err()$
$Cust.Trip\_Paid(CsI, fg_i, ht_j, Cxt)$
Trip_Paid
TrpPaid(...)
$\langle AG|BkdLst : Bkd.[CsI.fg_i.ht_j], ArchvLst : AchvL\rangle$
$\langle AG|BkdLst : Bkd, ArchvLst : AchvL.[CsI.fg_i.ht_j]\rangle$
Ttrp_paid
True
TravelStart
TrpStrt(..)
$Cust.Start2Travel()$
$Trip\_Booking - Err()$
$Trip\_Confirm - Err()$
$Trip\_Choose - Err()$
$Trip\_Requst - Err()$
AgcOP_Err
TrvRqErr(Cs,...)
TrvBkErr(Cs,...)

Figure 5.2: Behavioural Choreographical Specification of Travel Agency Service

`Trip_Find`, we are deriving the transition `Ttrp_Rqd`. The conditions of this transition are exactly the different conditions from these business rules. The input arc-inscriptions are all the required messages and service state parts from the place `Agency_St`. For our transition `Ttrp_Rqd`, the first alternative concerns the request for just travel (i.e. (`TrInf.Typ ="TR"`) and thus only a flight. In this case, as stated in the business rule, if the date is valid, the flight belongs the privileged ones and its fare is less than the customer budget, a request for the flight is made with a discount of 10 percent. The second alternative concerns a complete trip, in which case both flight and hotel are requested. In the same spirit all the other transitions are constructed.

We should note again that each time messages are sent the other services such as the airlines, hotels, banks and so on, the corresponding intra-services business rules at the level of such services are to be applied. In this way all received messages such as `Ttrp_Found`, `Ttrp_Confrmd` are already certified with respect to such intra-services business rules.

5.5 Summary

Web standards propose a purely static and structural descriptions for the Web services choreography through the XML-based standards such as WSCI and WS-CDL. These standards, when dealing with complex composite services are thus well-known for their limitations for coping with behavioural, adaptive and/or concurrent features characterising today's complex services. This chapter concentrates on the *choreographical* composition of several participating (elementary or complex) services to produce new complex *composite* added-value services. Therefore, this chapter pushes the scope of the proposed approach one step further, so that complex *composite* services can be harmoniously tackled on top of already specified and validated individual services and interfaces using CSrv-Nets. Also, an adequate generic pattern for cross-organisational or inter-service event-driven business rules has been proposed.

Chapter 6

Extending CCSrv-Nets for Dynamic Adaptability

In the two previous chapters, we proposed a stepwise formal approach for developing knowledge-intensive adaptive composite services. The proposed approach harmoniously brings together both orchestration and choreography, and it captures knowledge-intensiveness through event-driven business rules both at the intra- and cross-service levels. The forwarded Service-based Petri Nets Formalism and its choreographical extension CCSrv-Nets allows for intrinsically and soundly integrating these intra- and cross-services two-level business rules. We should further recall that this service formalisation is adaptive by construction since event-driven- business rules are by nature adaptive and evolving as they reflect business policies and strategies of any organisation, service or alliance of services. Business rules allow thus for keeping such (cross-)organisations competitive and attractive [48]. Otherwise, any cross-organisations offering just rigid (i.e. non rule-centric) services as the case with Web service standards including WSDL, BPEL and WSCI which cannot attract customers or even survive in so highly-volatile and fierce competitive global markets and economy.

In the so far approach, we have thus been supporting adaptability through business rules, which we soundly integrate in the construction of the service-based Petri nets model. More precisely, in order to adapt any business rule such as those related to single-service like airlines or the cross-services like the agency within an already specified CCSrv-Nets, we must redesign the corresponding transition capturing such business rules. The redesign in this formalism means, to change any transition, we have to explicitly replace all its input/output inscriptions as well as its conditions with the associated ECA (events-conditions-actions) elements of the newly introduced business rule. In other words, the so far addressed rule-centric adaptivity is exclusively achieved at the *design-time*.

Before motivating and delving into the details about how to leverage such design-time rule-centric adaptability towards *dynamic runtime* adaptability, we judged it essential to recall some of the benefits of such design-time evolution, we have been coping with in the previous chapters. Firstly, as we detailed in the literature assessment, no existing approach has tackled business rules in service-driven applications at the rigorous conceptual-level (besides the semi-formal description) even at the design-time. Secondly, we have been coping with both intra- and cross-service business rules in a harmonious complementarity. Thirdly, due to the concurrent and visual capabilities of CCSrv-Nets, business rules governing different activities/transitions behaviour can be simultaneously checked and executed. Fourthly, while changing a given business rule governing a transition behaviour, all other activities/transitions may be kept running; even the transition under design-time change can be executed with respect to the current business rule (i.e. before confirming the new one).

The purpose of the chapter aims above all at keeping all these benefits, while leveraging the adaptability of any business rules from design-time to a fully *dynamic runtime* adaptability. In other words, instead of blocking any transition(s) for explicitly updating their governing business rule, we will demonstrate how to achieve such adaptability on the fly, and with respect to any number of transitions. That is, while keeping the whole conceived CCSrv-Nets running, we perform the change in a dynamic and non-intrusive manner, so that even the concerned customer/user becomes unaware. Before presenting the main global ideas of this runtime adaptability as a smooth yet sound extension to the proposed design-time evolution, we present some of its basic principles, potentials and advantages.

Explicit Separation of Adaptability-level from a Conceptual One:

Achieving runtime, unanticipated adaptability automatically requires, on the one hand, an explicit separation between the running CCSrv-Nets conceptual-level under current business rules and the *adaptive-level*[1], where all past, present and coming rules have to be managed (e.g. removed, updated or added). On the other hand, adaptability implies the ability to dynamically weave/unweave (i.e. incorporate/extract) any business rule on the running CCSrv-Nets conceptual model. Indeed, with the so-far design-time rule-intensive service development using the CCSrv-Nets framework, the business rules are inherently *tightly-coupled* to their corresponding transitions; a fact which makes it impossible, among others, to supervise how the service rules are evolving along the

[1]In the literature we found different terminology for such explicit separation, including meta- [101, 17], reflection-level [62] or more recently aspectual-level. We instead prefer to use adaptive-level to expressively emphasise the purpose of that level

life-span of the service.

Explicit Focus on Rules and Their Evolution: As direct result of the clean separation of business rules from the (running composite CCSrv-Nets) service conceptualisation, corresponding (cross-)organisations become able to focus more on managing adaptability and enhancing competitiveness. In other words, business people become exclusively responsible for coping with policies and strategies in terms of business rules, whereas software designers will focus on how to formalise, implement and dynamically integrate such innovative rules.

Competitiveness Through Agility: Of course as we just emphasised, through the endowing of our conceptual model with an extra adaptability-level for business rules, we are promoting services to respond to any market changes, new emerging competitive policies, and/or new (rule-driven) attractive composition or opportunistic alliance with other services.

Personalisation of Rules to Specific Customers and Context: Besides adapting rules due to emerging policies and market changes, dynamic adaptability also directly facilitates the personalisation to specific instances of customers/providers and of times and locations as a simple case of context. More precisely, since we are promoting a "type-instance" conceptual model with High-level Petri nets, we are in position to incorporate more than one business rule for a given transition. Each business rule will be instantiate to the specific customer instances (e.g. golden, silver customers or providers). The same can be applied to the time, either in terms of specific days (week days and week-ends) or months (summer/winter sales and discounts). Finally, the location, where the customer of the provider is located or moving to can be captured through specific rules to be dynamically applied.

The remaining sections of this chapter are organised as follows. In the next section we informally motivate and present the main ideas and principles of the aimed dynamic adaptability in CCSrv-Nets. In the second section we illustrate these conceptualisation ideas on the already airline service CSrv-Nets specification, which becomes rule-centric and dynamically adaptable.

We conclude this chapter by outlining some of the very recent attempts and proposals to this challenging adaptability problem within service-oriented applications and Web services, and emphasise the advantages of the forwarded proposal.

6.1 Runtime Adaptability in CCSrv-Nets: Principles

First it is important to point out that using CCSrv-Nets structural and behavioural features, it is quite straightforward to add at any time *new* messages and new properties with associated business rules and construct their respective transitions, and this without any need to stop the running CCSrv-Nets model. Moreover, we can update any business rule ingredients (i.e. events/conditions/actions) by updating the arc inscriptions and the condition of the corresponding transition. However, what goes beyond the hitherto CSrv-Nets conceptualisation is the ability of dynamically manipulating any existing business rule governing a given transition. Beyond the CCSrv-Nets capabilities which belong further to the inability of dynamically endowing a given transition with more than one business rule, each acting for example on specific service instances.

In the following we progressively present our main ideas and principles to smoothly leverage CCSrv-Nets so that it can address such dynamic manipulation of transition behaviours,inherently governed through event-driven business rules.

6.1.1 CCSrv-Nets **Transitions (Meta-)*Representation***

Since any business rule governing an individual CSrv-Nets transition, the first question to come into mind towards manipulating such business rules, can be formulated as follows:

*"Is it possible to come up with an appropriate (meta-)*representation *for explicitly capturing any individual transition behaviour"*

Towards proposing an adequate transition's representation and thereby answering that question, we should first understand what are the ingredients comprising any CCSrv-Nets transition in the most general case. As a systematic response, a transition is definitely composed of: (1) a transition label or name; (2) input arc inscriptions with their corresponding input place names; (3) output arc inscriptions with their corresponding output places; and finally (4) the transition condition.

Consequently, a simple possible candidate representation of any transition may consist in *gathering* these four elements into a single *tuple*. Besides that, since we are aiming at changes with possibly several "versions" of such individual transition behaviour as tuple, we judge beneficial to enrich such tuple with a fifth element, as a natural counter reflecting the version identity of any specific behaviour related to a given transition. More precisely, we suggest a straightforward "user-friendly" transition's representation consisting of a five-element tuple of the form:

$$\langle\ \texttt{Transition\_identifier : current\_version | Input\_inscriptions,}$$
$$\texttt{output\_inscriptions, conditions}\rangle$$

Recalling again that Input (resp. Output)_inscriptions consist of all pairs "place with associated arc-inscription" getting in (resp. out) that transition, identified through the label "Transition_identifier".

To bring this "abstract" representation one step closer to our specific CCSrv-Nets framework, let us first (re-)conceive a generic pattern for CCSrv-Nets transitions. As depicted in 6.1, the most general form of transitions with respect to CSrv-Nets allow for bringing into contact different imported messages (and events denoted by Ms_{i_j}) with targeted service instance states (denoted by $\underset{i=1}{\overset{k}{--}}\langle Sid_i|prs_i\rangle$). Under specific conditions on both involved input message parameters and selected states properties, the transition general effect corresponds to the consumption of imported messages, the emerging of some new exported messages (denoted by Ms_{o_k}) and changes of some properties of the involved service states.

The instantiation of the above general transition's representation as a five-element tuple on this specific CSrv-Nets generic transition, results thus in the following more concrete five-element tuple.

$$\langle Tgnr : v \mid \underset{p=1}{\overset{n}{--}}(Msg_p, Ms_{i_p})\ (StSrv, \underset{i=1}{\overset{k}{--}}\langle Sid_i|prs_i\rangle)\ ,$$
$$\underset{j=1}{--}(StSrv, \underset{j=1}{\overset{k}{--}}\langle Sid_j|prs_j\rangle)\ \underset{q=1}{\overset{m}{--}}((Msg_q, Ms_{o_q})\ , TC(Tgnr)\rangle$$

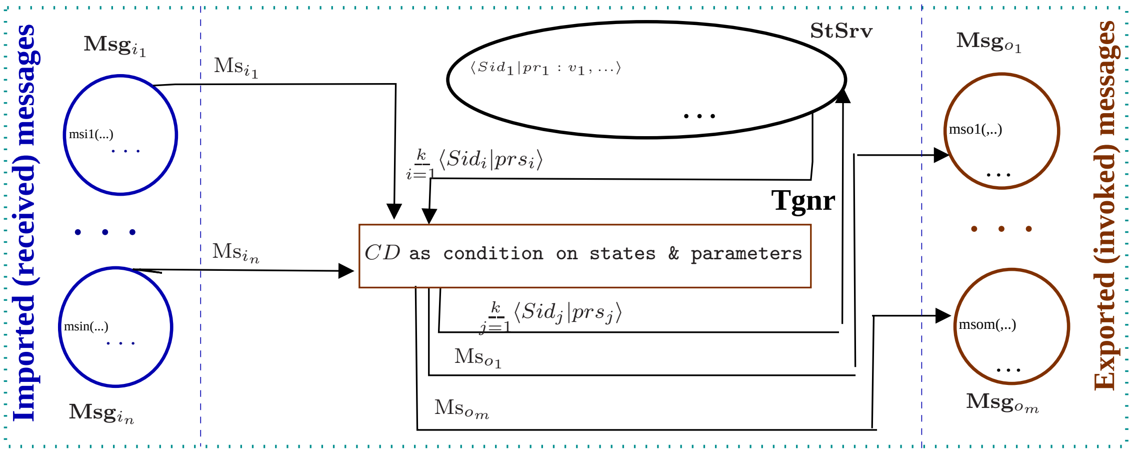

Imported (received) messages
Msg$_{i_1}$
msi1(...)
Ms$_{i_1}$
Ms$_{i_n}$
msin(...)
Msg$_{i_n}$
StSrv
$\langle Sid_1|pr_1 : v_1, \ldots\rangle$
$\sum_{i=1}^{k} \langle Sid_i|prs_i\rangle$
Tgnr
CD as condition on states & parameters
$\sum_{j=1}^{k} \langle Sid_j|prs_j\rangle$
Ms$_{o_1}$
Ms$_{o_m}$
Msg$_{o_1}$
mso1(,..)
msom(,..)
Msg$_{o_m}$
Exported (invoked) messages

Figure 6.1: The general pattern of CSrv-Nets transitions

where:

- $Tgnr$ represents a transition label or identifier. As we are aiming to update such transition as tuple, the corresponding name or label should of course be any specific label from existing CSrv-Nets transitions.

- v as we motivated should refer to a given specific version of such transition. By convention, we associate the counter zero (0) to the first behaviour. Any other adaptations will be referenced by incrementing v by one (1). We are thus using the version numbers to keep *track* of different changes.

- The third element of the tuple $\sum_{p=1}^{n}(Msg_p, Ms_{i_p})$ $(StSrv, \sum_{i=1}^{k}\langle Sid_i|prs_i\rangle)$ defines the different input messages and state places with their corresponding (multiset of terms) input arc inscriptions as given in the generic transition in Figure 6.1. Important to notice here is that such elements may contain *variables*, exactly like inscriptions associated with usual transitions. They are not like usual tokens which must be ground terms, that is, without any variable inside.

- The fourth element $\underset{j=1}{\dashv\!\vdash}(StSrv, \underset{j=1}{\overset{k}{\dashv\!\vdash}}\langle Sid_j|prs_j\rangle) \underset{q=1}{\overset{m}{\dashv\!\vdash}}((Msg_q, Ms_{o_q})$ captures the different output message and state places with their associated arc-inscriptions.

- Finally, the fifth element $TC(Tgnr)$ represents the (transition) condition we may associate with a given transition.

▶ **Example 6.1.1** Let us reconsider the airline CSrv-Nets from chapter four, that we reproduce here again in Figure 6.2 to spare the reader time looking for it. Let us focus, for instance, on the transition `Flight_Book`. Following the above generic transition's representation and its CSrv-Nets instantiation, the five-element tuple governing the transition `Flight_Book` in the Airline CSrv-Nets specification, could be represented as follows:

$\langle Tflight\_bk : 0 \mid (Flight\_Book, FlgBk(Cs, R, Dy, Py))(Fligh\_St,$
$\langle FG|FgInf : [R.Fr.To.Dt.Tm.Cx], Rsv.Rs, Cmf : Fm\rangle)$ **,**
$(Flight\_Bookd, FlgBkd(Cs, R.Fr.To.Dt.Tm, Py)(Flight\_Pay,$
$FlgPay(Cs, R, Py) \wedge FlgPnl(Cs, R, Pn)$ **,** $(Dc \leq Dy) \wedge (Cs \in Rs) \wedge (Py = Cx) \wedge (Pn = 0) \wedge (Cmf.[Cs.R]) \wedge ((Dc \geq Dy) \wedge (Pn := Py * 0.1))\rangle$

That is, first since this transition behaviour is the default one we assigned it the version zero (0). For both input (resp. output) arc-inscriptions, we associated to them their corresponding input (resp. output) places. For instance, we are coupling the input place `Flight_Book` with its corresponding arc-inscription, that is, $FlgBk(Cs, R, Dy, Py)$. The same process is applied to all other (input and output) places. Finally, the fifth element in the tuple is the condition, where we have simply skipped the `Else` part.

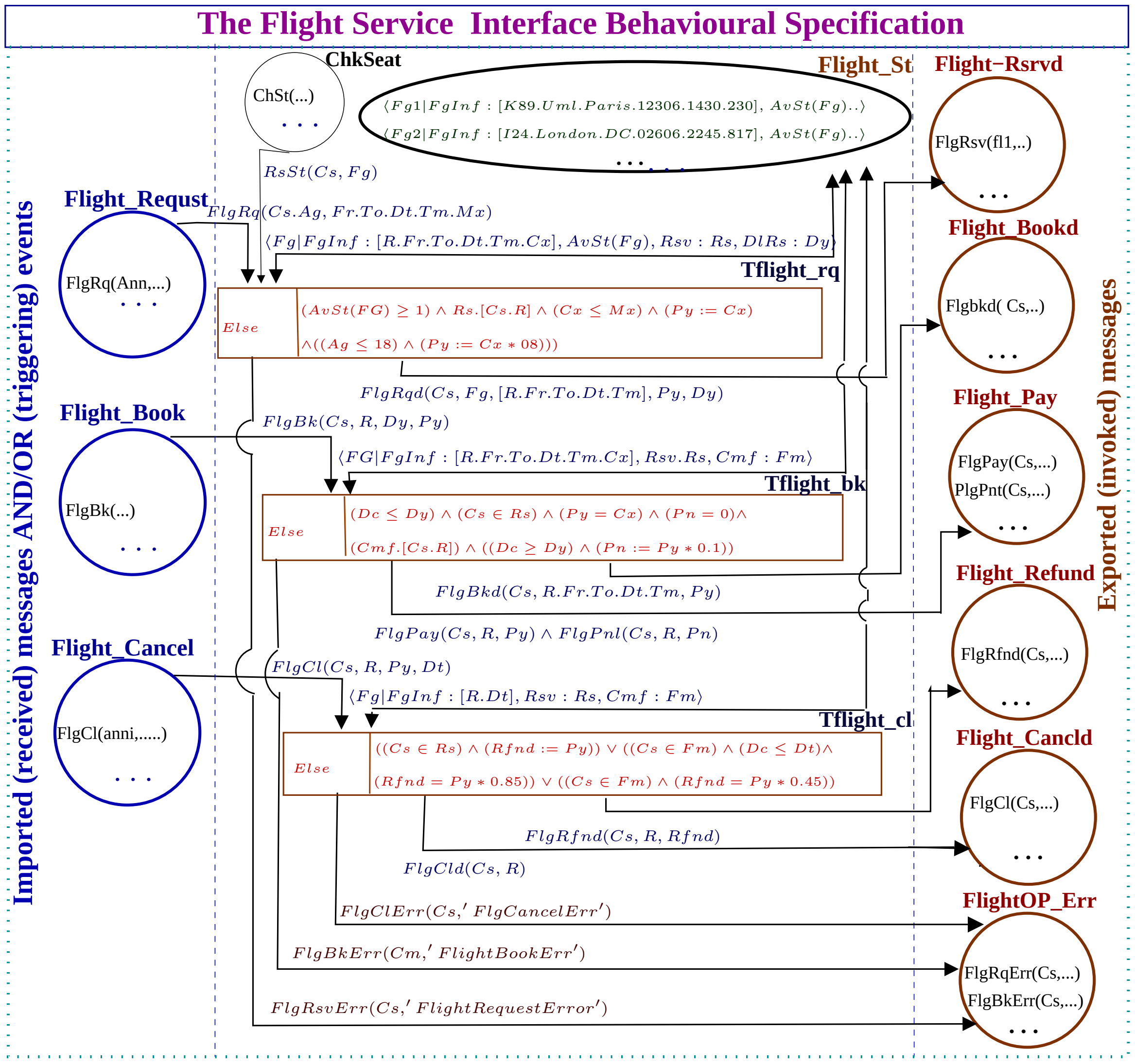

The Flight Service Interface Behavioural Specification
ChkSeat
ChSt(...)
Flight_St
$\langle Fg1|FgInf : [K89.Uml.Paris.12306.1430.230], AvSt(Fg)..\rangle$
$\langle Fg2|FgInf : [I24.London.DC.02606.2245.817], AvSt(Fg)..\rangle$
Flight−Rsrvd
FlgRsv(fl1,..)
$RsSt(Cs, Fg)$
Flight_Requst
FlgRq(Ann,...)
$FlgRq(Cs.Ag, Fr.To.Dt.Tm.Mx)$
$\langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy\rangle$
Tflight_rq
$Else$
$(AvSt(FG) \geq 1) \wedge Rs.[Cs.R] \wedge (Cx \leq Mx) \wedge (Py := Cx)$
$\wedge((Ag \leq 18) \wedge (Py := Cx * 08)))$
Flight_Bookd
Flgbkd(Cs,..)
$FlgRqd(Cs, Fg, [R.Fr.To.Dt.Tm], Py, Dy)$
Flight_Book
FlgBk(...)
$FlgBk(Cs, R, Dy, Py)$
$\langle FG|FgInf : [R.Fr.To.Dt.Tm.Cx], Rsv.Rs, Cmf : Fm\rangle$
Tflight_bk
Flight_Pay
FlgPay(Cs,...)
PlgPnt(Cs,...)
$Else$
$(Dc \leq Dy) \wedge (Cs \in Rs) \wedge (Py = Cx) \wedge (Pn = 0)\wedge$
$(Cmf.[Cs.R]) \wedge ((Dc \geq Dy) \wedge (Pn := Py * 0.1))$
$FlgBkd(Cs, R.Fr.To.Dt.Tm, Py)$
Flight_Refund
FlgRfnd(Cs,...)
$FlgPay(Cs, R, Py) \wedge FlgPnl(Cs, R, Pn)$
Flight_Cancel
FlgCl(anni,.....)
$FlgCl(Cs, R, Py, Dt)$
$\langle Fg|FgInf : [R.Dt], Rsv : Rs, Cmf : Fm\rangle$
Tflight_cl
$Else$
$((Cs \in Rs) \wedge (Rfnd := Py)) \vee ((Cs \in Fm) \wedge (Dc \leq Dt)\wedge$
$(Rfnd = Py * 0.85)) \vee ((Cs \in Fm) \wedge (Rfnd = Py * 0.45))$
Flight_Cancld
FlgCl(Cs,...)
$FlgRfnd(Cs, R, Rfnd)$
$FlgCld(Cs, R)$
FlightOP_Err
$FlgClErr(Cs,' FlgCancelErr')$
$FlgBkErr(Cm,' FlightBookErr')$
FlgRqErr(Cs,...)
FlgBkErr(Cs,...)
$FlgRsvErr(Cs,' FlightRequestError')$
Imported (received) messages AND/OR (triggering) events
Exported (invoked) messages

Figure 6.2: The Behavioural Specification of Flight Service Interface.

6.1.2 Towards PN-Based *Manipulation* of Transition-Behaviours as-Tuple

After being able to capture any transition-behaviour as tuple in a given CSrv-Nets conceptual model, the next decisive step concerns the independent and dynamic manipulation (e.g. updating, adding and removing) of such tuples.

Moreover, to stay compatible with our CSrv-Nets framework, such manipulation should be geared and based on (high-level) Petri nets; otherwise it would impossible to dynamically link the CSrv-Nets with the envisioned adaptability level. More precisely, the Petri nets-based proposal we are looking for to realise such manipulation, could be summarised in the following steps:

(1) At first we propose to *gather* such (transition-behaviours) tuples into an associated **(meta-)place**. Such place which should reflect the first construct of the expected adaptability level allows thus for keeping all past and current business rules.

(2) Given such tuples within that meta-place, we require then further (meta-)places and transitions for effectively manipulating them.

Places for Changes Triggering: Since we are looking for updating/modifying, adding and/or removing any tuple, we propose three corresponding (meta-operation) places, we denote by **Chg-BhvTp**, **Add-BhvTp** and **Del-BhvTp**. These ((upper-)places permit thus for storing any corresponding (meta-)messages for changing, adding or removing any tuple. As will be detailed later, we may denote such (meta-)messages with respectively `Chg(beh-as-tuple)`, `Add(beh-as-tuple)` and `Del(beh-as-tuple)`.

Transitions for Manipulating Tuples: Given such (meta-operation) places and the tuple place, the last step for effectively enabling any manipulation consists in conceiving three associated (upper-)transitions. By denoting such transitions for instance as T_{ChgBh}, T_{AdBh} and T_{DlBh}, they will respectively relate **Chg_Bh**, **Ad_Bh**

and **Dl_Bh** places to the **(meta- or upper-)place**. Such transitions permit thus selecting any (meta-)message instance as token from the corresponding meta-message place and associate to it the existing tuple (except for addition of a tuple) from the meta-place marking and perform the required operation (i.e. addition, removal or update) on that tuple.

The upper-layer as illustrated in Figure 6.3 exactly reflects these explained principles towards dynamic adaptability of the transition's behaviour as-tuple. More precisely, this proposed general upper adaptability-level is again a specific variant of high-level Petri nets, but with tokens including variables as they capture a complete transition behaviour. Note that for adding (transition-)behaviours we have indeed conceived two transitions, instead of just one. The reason is that while adding a new behaviour of a given base-level transition, two cases are to be distinguished. The first case concerns the situation where the to-be behaviour corresponds to a completely new transition; that is such transition does not already exist in the meta-place (we use the symbol ~ to check this non-existence i.e. as inhibitor arc). In this case the version is set to one (1). The second case concerns the adding of the new version behaviour to an existing transition, where we have to increment to the version counter.

6.1.3 Connecting the Adaptability-Level to the Conceptual Base-Level

So far we motivated and presented how to conceive the adaptability-level as a variant of high-level Petri nets. The next crucial step concerns the connection of this adaptability-level to the running base-level CSrv-Nets conceptualisation.

Once such connection is established we have to finally develop on how to perform the weaving/unweaving (or shifting-down/shifting-up) of any new/existing transition behaviour as token; that is how to dynamically (dis-)activate any event-driven business rule.

Since the meta-place at the adaptability-level contains tokens as five-element tuples, which we aiming at propagating as a normal CSRV-NETS transition at the base conceptual-level, the appropriate necessary linking consists of the two essential following mechanisms:

***Enrich* all Conceptual-level Transitions with Extra (meta-)*variables*:**

The first mechanism for preparing the runtime propagation/extraction of any selected meta-token (i.e. business rule from the meta-place) consists of *slightly upgrading* transitions of the concerned (and *already* conceived and validated) CSRV-NETS conceptual model with some extra *meta-variables*. We are using the term meta-variables, because such variables will be possibly instantiated with terms containing variables. In other words, we are (almost) keeping the conceptual-level as it is except for such additional extra (meta-variables). More precisely, using a new construct we denote by $\bowtie$ we enrich all input/output inscriptions and conditions of any transition with such meta-variables. The new summing-up operator $\bowtie$ as will semantically be explained later, plays the role of both (*wedge*) and or (*vee*) depending on the aimed propagation strategy. That is, either we want to add the associated inscription to an existing behaviour or substitute that inscription with a new one. As an illustration, we are upgrading the input-arc inscription for the input message place $\mathtt{Msg}_{i_1}$, from $\mathtt{Ms}_{i_1}$ to the enriched inscription $\mathtt{Ms}_{i_1} \bowtie IT^m_{i_1}$. With respect to the generic CSRV-NETS transitions pattern as depicted in the lower level of Figure 6.3, these meta-variables and their roles could be summarised as follows:

- The meta-variable IT^s (resp. IT^m) permits receiving any *input*-inscription from the given selected meta-token related to the service state (resp. imported messages). That is, IT is referring to "input-(meta-)token", whereas s (resp. m) is referring to service state (resp. imported message).
- The meta-variable OT^s (resp. OT^m) permits for receiving any *output*-inscription from the given selected meta-token related to the service state (resp. exported messages).
- Finally CD allows for receiving the condition part of any selected meta-token.

Red-arcs Between the Meta-place and Any CSrv-Nets Transitions: To permit bringing down/up any transition-behaviour as token from the meta-place to any upgraded transition in the CSrv-Nets base-level, we require to connect them in both directions, using thus input and output arcs as red-arcs. For that purpose, we should use the already brought in "meta-"variables to the concerned transition.

Taking these meta-variables into consideration, as shown in the middle of the Figure 6.3, the resulting arc-inscription relating the meta-place to any base-level CSrv-Nets transition takes the following form :

$\langle Tgnr : v \mid (StSrv, IT^s) \frac{i_p}{i=i_1} (Msg_i, IT^m_i) \, , \, (StSvr, OT^s) \frac{h_r}{j=h_1} (Msg_j, OT^m_j), CD^t \rangle$

That is, besides the transition label and a variable for keeping track of the version, all input (resp.) output places are paired with only the respective *meta-variables* from associated arc-inscriptions. As we will detail later on, the already defined initial behaviour (i.e. the arc-inscriptions and condition without these variables), though displayed on the transition as "referential" initial behaviour, in case of dynamic changes will not come into play.

6.1.4 Mechanisms for Shifting Down/Up Business Rules

The last phase in this advanced conceptualisation towards addressing (rule-driven) runtime adaptability in any CSrv-Nets service specification, consists in precisely clarifying how the smoothly enriched CSrv-Nets transitions will be executed or fired. In other words, how should we *dynamically* at runtime bring down (and up) any transition-behaviour as meta-token from the meta-place and execute it at the associated CSrv-Nets base-level transition. In the following we thus informally explain this process, and afterwards define it in a more disciplined manner.

Let us recall again the meaning of the crucial operator $\bowtie$ relating any already existing initial arc-inscription with to be superposed on woven non-instantiated inscription (as meta-variable). To cover most of adaptability (and non-adaptability) cases, we propose to assign to this $\bowtie$ operator the following four semantical interpretations:

meta-var **as nil:** The fact of enriching any specified CSrv-Nets transition with just extra meta-variables should *obligatory* imply that all transitions have to be adaptable. Indeed, in real-word service applications, there are always some business activities (as transitions in our model) which require to be fixed forever. For that reason, we are allowing in the proposed interpretation to interpret, in such case, all extra inscriptions " `meta-var`" as if they do not exist, i.e. as *nil*. In other words, we are free to make any CSrv-Nets transition as evolving or fixed, depending on the real-world application at hand. Of course, any fixed transitions can be shifted at any time towards evolving ones by interpreting the extra enriching part (i.e. " `meta-var`") following the next alternatives.

meta-var **as or ($\vee$) operator:** Interpreting the operator as a choice $\vee$ operator means that for firing the concerned transition, we should dynamically bring down a new behaviour from the meta-place. In other words, the old existing initial behaviour is to be skipped and dynamically replaced by any wished new business rule (as meta-token). In this case, the old behaviour is *completely* swaped with the new propagated one, and its presence at the arc-inscriptions permit just to remind the designer of that initial behaviour.

meta-var **as and ($\wedge$) operator:** The last possibility, we propose to offer while dynamically bringing down any behaviour as-token, consists in dynamically softening/ tightening/enriching any already existing behaviour with more knowledge. That is, we propose to interpret the operator as a conjunction, which means integrating the newly woven behaviour with the existing one. For instance adding a new behaviour at the condition means tightening further that condition, adding new input/output messages while involving an existing transition means bringing in more flexibility.

Mixing the operator as any of the three: Although we will not detail it further, from the above interpretations it is quite straightforward to consider *within a same transition* all the three possibilities with respect to selected arc-inscriptions. For instance, we may skip adapting some input messages, while dynamically tightening via as an add ($\wedge$) and softening some output messages by interpreting the operator as an ($\vee$). As a convention in the case where nothing is to be changed within a given arc-inscription, we propose to use the minus symbol "-" within the related position in the corresponding meta-arc. For instance, for the following meta-arc $\langle Tg1 : 1 \mid -, \ -, \ CD^t \rangle$, we mean that all input and output inscriptions remain unchanged except the condition, which can be adapted.

In the following, we go into detail about one of the four above possibilities, namely the choice case. We further highlight how the other cases (particularly the third and the fourth) can be adjusted following the forwarded weaving conceptualisation.

To confirm the smooth and progressive shifting from the CSrv-Nets base-level conceptualisation towards this dynamic adaptability, we would like to recall and shortly report on the firing semantics we already introduced and discussed in chapter 4. The semantics of CSrv-Nets transition firing has been formalised through the definition 6.1.2, we again recall hereafter:

Definition 6.1.2 (CSrv-Nets-transition semantics) We assume a given marked CSrv-Nets net, with its marking state denoted by $\mathcal{M}_{st} = \underset{k}{-}(p_k, M(p_k))$. Further, we assume as above that transitions are represented as a five-element tuple but for sake of simplicity and understandability we are using instead the place names given in Figure 6.1. That is, instead of Msg_i, $StSrv$), we are using p_i and mt_i (resp. q_j and nt_j) for referring input message and state (resp. output) places.

$$\langle t : v \mid \underset{i}{--}(p_i, mt_i), \underset{j}{--}(q_j, nt_j), [TC(x)]_{bool} \rangle$$

The firing conditions and outputs are formally expressed through the following inference rule. That is, we have first to find a set of substitutions $\{\sigma_i\}$ for transition variables with corresponding terms from associated ground terms $T_{p_i}(\emptyset)$ of same sort. The main condition is that such ground terms should belong to the corresponding input message/state places. Moreover, with respect to the union of such found substitutions (denoted by σ), the transition condition for all variables appearing in the condition $\sigma([TC(x)]_{bool})$ should hold true. The resulting marking (i.e. conclusion part of this firing inference rule) consists in such case in taking away all the matched tokens from the input places and produce the new resulting tokens to corresponding output places (i.e. $\sigma(\underset{j}{--}(q_j, \sigma(nt_j))))$.

$$\frac{\exists\, \sigma_i : x(t) \rightarrow [T_{p_i}(\emptyset)] \mid \underset{i}{--}(p_i, \sigma(mt_i)) \in \mathcal{M}_{st} \wedge (\sigma([TC(x)]_{bool}) = True)}{\mathcal{M}'_{st} = \mathcal{M}_{st} - \underset{i}{--}(p_i, \sigma(mt_i)) + \sigma(\underset{j}{--}(q_j, \sigma(nt_j)))}$$

Obviously this firing mechanism as such *cannot* be applied to our newly enriched transitions, i.e. to our "adaptability-aware" or simply adaptable transitions as depicted in the low-part of Figure 6.1. The main challenging emerging problem concerns thus the propagation of meta-token through the red-arc relating the (adaptability and base) two-levels. To resolve this problem and allow firing such enriched transitions, we propose instead a *two-phased instantiations*. That is, in order to execute such adaptable transitions we first require to dynamically select the right "behaviour as meta-token" from the meta-place of the adaptability-level. Secondly, and only after that we will be in a position to fire that newly and dynamically propagated behaviour following the above usual mechanism.

To be more concrete the dynamic selection of a given behaviour as meta-token (i.e. as an event-driven business rules) formally returns in finding a (meta-)substitution we may denote as σ^{vr}, which permits instantiating *all* the involved meta-variables, that are IT^s, IT^m, OT^s, OT^m and CD, so that they exactly match one of the behaviour as meta-token in the meta-place. Because such behaviour-as-meta-token can be dynamically manipulated through the developed adaptability-level, we are thus realising a runtime adaptability of business rules. Note that we are using the notion of meta-substitution for σ^{vr}, due to the fact that this substitution in contrast to those adopted $\{\sigma_i\}$ in the above definition instantiates the (meta-)variables with corresponding terms *with variables*.

Once selecting such specific transition behaviour from the meta-place, the next step consists of firing that transition exactly as we defined in the above definition. In terms of the precise inference rule, this two-phase execution of adaptable transitions can be formulated as follows.

Definition 6.1.3 (adaptable CSrv-Nets**-transition semantics)** We assume given a marked adaptable[2] CSrv-Nets model directly connected with an adaptable-level "meta-CSrv-Nets" marked net as we developed. We denote the base-level CSrv-Nets marking state by $\mathcal{M}_{st} = \underset{k}{--}(p_k, M(p_k))$. The marking of the meta-place of the adaptable-level framework is denoted as follows $\mathcal{M}(p_{meta})$.

The base-level adaptable generic transition, as given in Figure 6.3 takes the

[2]With enriched transitions by meta-variables though the operator .

following form. Note that we are using the the abbreviations p_i and q_j instead of Msg_i and $StSrv$ and abstracting IT^s and IT^m into a common IT meta-variable and OT^s and OT^m into OT—, takes the form:

$$\langle Tgnr : v \mid \underset{i}{-\!-}(p_i, mt_i \bowtie IT_i), \underset{j}{-\!-}(q_j, nt_j \bowtie OT_j), [TC(X)]_{bool} \bowtie CD_{Tgnr} \rangle$$

In the above explanation and the definition 6.1.2, the inference rule allowing for firing this adaptable transition can be expressed as follows.

$$\exists\, \{\sigma^{vr}\} : X(Tgnr) \rightarrow [T_{P_{meta}}(X)] \mid$$

$$\langle Tgnr : \sigma^{vr}(v) \mid \underset{i}{-\!-}(p_i, \sigma_i^{vr}(IT_i)), \underset{j}{-\!-}(q_j, \sigma_j^{vr}(OT_j)), \sigma^{vr}(CD(Tgnr)) \rangle \in \mathcal{M}(p_{meta})$$

$$\frac{\exists\, \sigma_i : x(t) \rightarrow [T_{p_i}(\emptyset)] \mid \underset{i}{-\!-}(p_i, \sigma(\sigma_i^{vr}(IT_i))) \in \mathcal{M}_{st} \wedge (\sigma(\sigma^{vr}(CD(Tgnr))) = True)}{\mathcal{M'}_{st} = \mathcal{M}_{st} - \underset{i}{-\!-}(p_i, \sigma(\sigma(\sigma_i^{vr}(IT_i))) + \sigma(\underset{j}{-\!-}(q_j, \sigma(\sigma_j^{vr}(OT_j)))))}$$

To mix any knowledge in already existing or default transition behaviour with the dynamically propagated behaviour, the above firing mechanism could be straightforwardly reshaped by including the (instantiated) arc-inscriptions in the second phase of the instantiation process. That is, we will not involve just IT, OT and CT as we did above, but also ms_i, ms_j and $TC(x)$ while substituting for the final firing of the transition.

6.2 A Dynamically Adaptable CSrv-Nets Airline Specification

The objective of this section is to apply the above conceptualisation for dynamically adapting the airline CSrv-Nets specification, we already detailed in chapter four and recalled in this chapter through Figure 6.2. To illustrate this dynamic adaptability of that airline specification, we require to progressively put into force the above mechanisms. More precisely, first we have to endow any transition of this Airline CSrv-Nets specification, with appropriate (meta-)variables so that they become adaptive-aware. As a second step, we have to built the adaptability-level, within which any business rule can be dynamically manipulated. Finally, we show how any of these dynamically manipulated business rules can be (dis-)activated by (un)weaving it to the base-level.

6.2.1 Upgrading the CSrv-Nets Airline Towards Adaptability-Awareness

As we pointed out the first step towards endowing any knowledge service-driven CSrv-Nets specification consists in preparing that specification to become adaptable-aware. More precisely, for each of the CSrv-Nets airline transitions, we have to slightly enrich its (input/output) arc-inscription as well as the condition part with a (meta-)variable using the operator $\bowtie$.

The result of applying this enrichment is depicted in the low-level of Figure 6.3. First note that to ease the manipulation, instead of the long name for places (and transitions) we are shortening them. So, for instance, instead of the place name `Flight_Book`, we are using just `FlBk`. Second, because we want that all activities of the flight service to be adaptable, we are enriching all the three transitions; again here for the sake of simplicity we are dropping the `Else` part (i.e. the exception cases) in all these transitions.

6.2.2 Building and Illustrating the CSrv-Nets Airline Adaptability-Level

Towards effectively bringing this runtime knowledge-centric adaptability conceptual machinery to the above airline CSrv-Nets prepared specification, let us first consider three simplified business rule scenarios: Two concerning the booking request business activity (i.e. the transition `Tflg_rq`) and one new business rule dealing with the cancelling activity (ie. the transition `Tflg_cl`). Of course as will be understood from this progressive illustration, we will be able to dynamically deal with any business rule and with respect to any business activity as transition.

These three business rules could informally be described as follows:

Flight to A Specific Destination and Period (`R1`): This rule says, for instance, that any person travelling to Cairo or Istanbul between June and August gets a discount of 50 percent from the normal fare.

Flight to A specific Destination for A Group (`R2`): Any two persons travelling for instance to Las Vegas during the month of January will automatically get a 30 percent discount.

Seasonal Flight-cancel (`R3`): This rule stipulates that during the winter (Christmas time) season a refund will correspond to the VAT, whereas during the summer the refund concerns only half of the paid price.

The next step after this informal description of any business rules to be dynamically integrated in the running Airline CSrv-Nets specification consists in *formally* expressing it as a five-element tuple with respect to the transition governing the associated business activity. In the following the translation of these three informal rules to their precise five-element tuples will be described.

The business rule `R1` as tuple. The first rule `R1` concerns the flight request business activity, that is to say, it concerns the transition `Tflg_rq` in the upgraded CSrv-Nets airline specification. As it is the first rule to be introduced in addition to the default rule, the counter for the version is to be set to one (1). Moreover, when analysing this business rule, we see it mainly brings new constraints or conditions. In other words, both the input and output places with their corresponding arc-inscriptions remain unchanged as it is given by the default behaviour (initial business rule). As we afore-suggested, in this case the second and third elements in the tuple have to be set to "-". Finally, using the variables from the (default) input messages and the input service state (i.e.), the described conditions are straightforwardly expressed into the following formal expression: $(Fr = \texttt{"Cairo"} \vee \texttt{"}Instanbul\texttt{"}) \wedge (\texttt{"June"} \leq Dt \leq \texttt{"August"})$ $\wedge(Py := Cx * .50)$

To summarise, the five-element tuple associated with the new business rule `R1` for flight request takes the following form:

$\langle Tflight\_rq : 1 \,|-, - ,(AvSt(FG) \geq 1) \wedge Rs.[Cs.R] \wedge (Cx \leq Mx)$
$\wedge\, (Fr = \texttt{"Cairo"} \vee \texttt{"}Instanbul\texttt{"}) \wedge (\texttt{"June"} \leq Dt \leq \texttt{"August"})$
$\wedge(Py := Cx * .50)\rangle$

The business rule `R2` as tuple. This rule also concerns the flight request and thus the transition `Tflg_rq`. This systematically means that it is the second version or alternative besides the default behaviour, and thus the counter for the version is to be set to two (2). In contrast to the first rule, this rule is to be triggered by the simultaneous occurrence of two requests (i.e. from two persons). This implies that from the input place `FlRq` (flight-request) we require two messages, one for instance from `Cs1` and the second from `Cs2` but the same parameters (i.e. origin, destination, date, cost). The third element in the tuple, that is, the output places and their corresponding inscriptions remain unchanged as in the default; so we abbreviate them using the symbol ”-”. The last element in the tuple concerns the condition, which was formulated for the above rule. We should just note that since two persons are at stake, the available seat should greater than 2 and both the two customer Ids (i.e. `Cs1` and `Cs2`) have to be added to the reservation list `Rs`. The flight cost should of course be less than the maximum budget of any of the two customers. All this tuple takes the following form:

$\langle Tflg\_rq : 2\ |(FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx)\ FlgRq(Cs2.Ag, Fr.To.Dt.Tm.Mx))\ (FlSt, \langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy\rangle)$ **,– ,** $(AvSt(FG) \geq 2) \wedge Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx) \wedge (Fr = \texttt{"Las Vigas"}) \wedge (\texttt{"Jan."} = Dt) \wedge (Py := 2 * Cx * .70)\rangle$

The business rule `R3` as tuple. This new business rule concerns the cancel activity and thus the transition `Tflg_cl` as the first version besides the default version. As for the first introduced rule, this rule mainly focusses on the condition part, and henceforth the two input and output of the second and third elements of the tuple as abbreviated to ”-”. The condition itself is composed of a disjunction of two expressions: one concerning the Christmas period (i.e. between 15 and 30th of December), where the sum to be refunded is to set the

VAT, and the summer period where just half is refunded. all the resulted tuple is to be expressed as follows:

$\langle Tflg\_cl : 1 \,|- ,- ,(Cx \in Fm) \wedge (\texttt{"15 Dec."} \leq Dt \leq \texttt{"30 Dec."})$
$\wedge(Rfnd := Vat(Py)) \vee (\texttt{"May"} \leq Dt \leq \texttt{"Aug."}) \wedge (Rfnd := Py * .5))\rangle$

6.2.3 The Emerging of the Three (Rules-As-)Tuples at the Adaptability-Level

As depicted in Figure 6.3, for simplicity we have skipped all the places and associated transitions for manipulating the rules as tuples. In other words, we just assume that the three above tuples have been introduced using the (meta-)transition T_{AdBh} (for the second rule T_{AdBh1} `as it is the second version`. That means that the place `Ad_Bh` should have been containing three tokens of the form:

- $Add\_Bh(Tflg\_rq, \_, \_, (Fr = \texttt{"Cairo"} \vee \texttt{"}Instanbul\text{”})$
  $\wedge(\texttt{"June"} \leq Dt \leq \texttt{"August"}) \wedge (Py := Cx * .50)$

- $Add\_Bh(Tflg\_rq, (FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx)$
  $FlgRq(Cs2.Ag.Fr.To.Dt.Tm.Mx)\ (FlSt, \langle Fg|FgInf \quad : \quad [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv \quad : \quad Rs, DlRs \quad : \quad Dy\rangle)\ ,- \ , (AvSt(FG) \geq 2) \wedge\ Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx)$
  $\wedge(Fr = \texttt{"Las Vigas"})\ \wedge(\texttt{"Jan."} = Dt) \wedge (Py := 2 * Cx * .70))$

- $Add\_Bh(Tflg\_cl, - \ ,- \ ,(Cx \in Fm) \wedge (\texttt{"15 Dec."} \leq Dt \leq \texttt{"30 Dec."})$
  $\wedge(Rfnd := Vat(Py))\ \vee(\texttt{"May"} \leq Dt \leq \texttt{"Aug."} \wedge (Rfnd := Py * .5))$

The firing of the (meta-)transition T_{AdBh} three times successively results in the emerging of three tuples in the meta-place `Brs.Airline-Place` as depicted in the upper-layer of Figure 6.3. That is to say, three new rules have been

dynamically introduced at that adaptability-level. To keep the Figure manageable we have indeed skipped this self-explained firing. Given these rules, it is important to emphasise that we can in the same spirit change and/or delete them through the two other (meta-) transitions (and meta-places).

6.2.4 The Dynamic Shifting Down/Up of these New Rules on the Base-Level

After demonstrating how any business rules can be manipulated at runtime at the adaptability-level, we will be able to activate any of these new rules for firing the corresponding transitions. This dynamic activation is to be governed using the two-step firing mechanism we forwarded in the above definition.

Besides this dynamic activation of any rule from the adaptability-level instead or in combination with the default initial behaviour, we can also at design-time change this default behaviour by any new emerging rule from the adaptability-level. In the following we illustrate this design-time update of default behaviours for both the flight request and cancel transitions. More precisely, let us bring the rule (`R1`) as a default for the transition `Tflg_rq` and the rule (`R3`) as a default for the transition `Tflg_cl`. This also means that the default behaviours of these two transitions have to be shifted up to the adaptability-level while shifting down those corresponding to (`R1`) and (`R3`).

Formally the shifting up consists of introducing the two default behaviours through the meta-transition `Tad_Bh` as we explained for the three emerging new rules. For the sake of simplicity, we are skipping this trivial phase and assuming directly that these two default behaviours have already been added to the meta-place through a two-time firing of this meta-transition.

The ultimate result is depicted in Figure 6.4. That is, first the default behaviour for both transitions `Tflg_rq` and `Tflg_cl` is shifted up at the adaptability-level. Secondly, the behaviour associated with the rules (`R1`) and (`R3`) is dispatched as it should be on the corresponding input/output inscriptions and conditions. In other words, with this judicious combination of design- and runtime-adaptability, we are in a position to dynamically adapt any CSrv-Nets specification and monitor that change.

6.3 Summary

The purpose of this chapter is to go beyond the design-time adaptability of behavioural features governed by event-driven business rules. This soundly extend the conceptual model by endowing its constituents (e.g. interfaces, local and global composite services) with adaptability level, where rules can be dynamically manipulated. For the dynamic shifting up/ down of such business rules of services, we propose reflection mechanism.

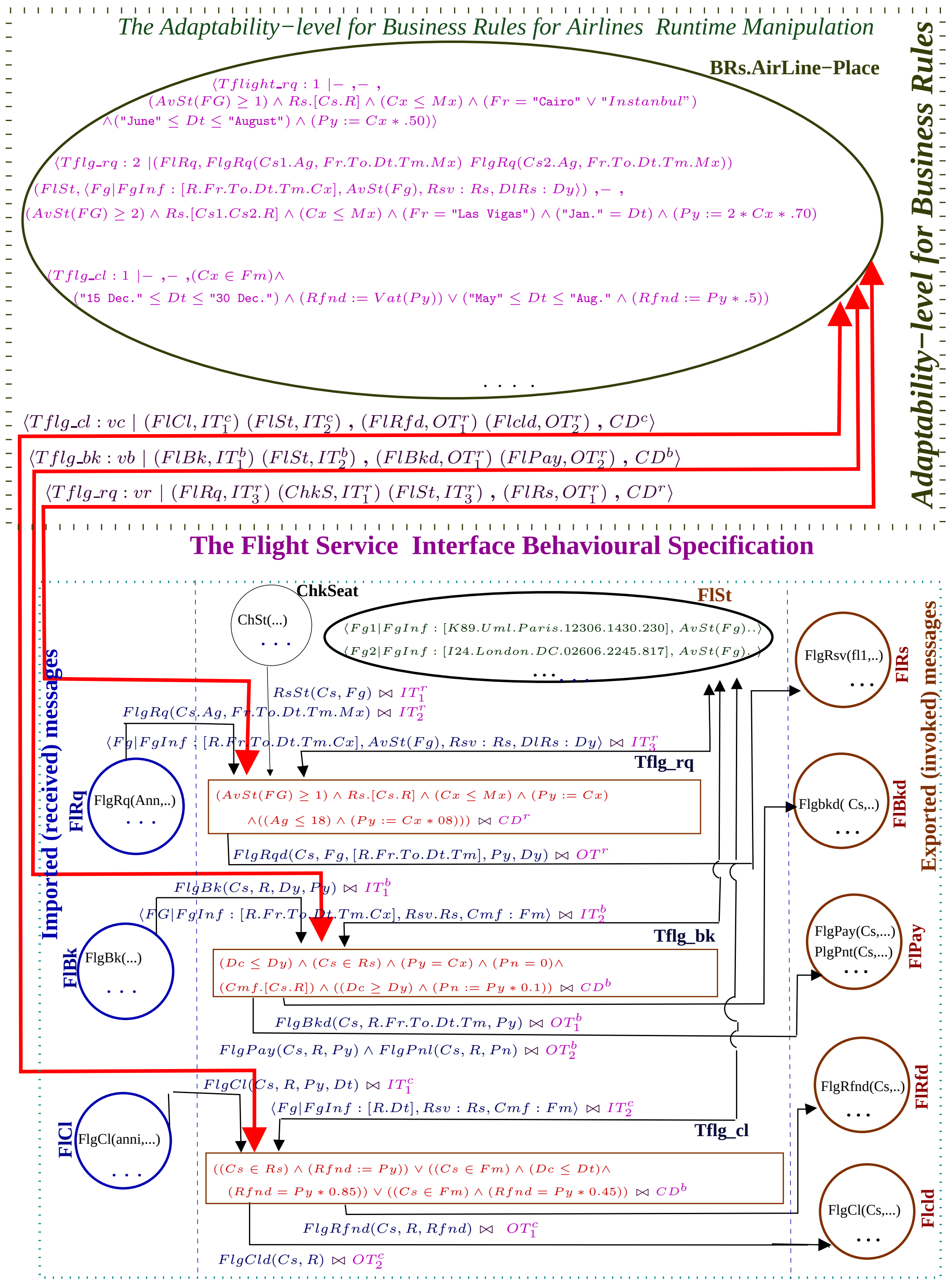

The Adaptability−level for Business Rules for Airlines Runtime Manipulation
BRs.AirLine−Place
Adaptability−level for Business Rules
$\langle Tflight\_rq : 1 \mid - ,- ,$
$(AvSt(FG) \geq 1) \wedge Rs.[Cs.R] \wedge (Cx \leq Mx) \wedge (Fr = \texttt{"Cairo"} \vee \texttt{"Instanbul"})$
$\wedge(\texttt{"June"} \leq Dt \leq \texttt{"August"}) \wedge (Py := Cx * .50)\rangle$
$\langle Tflg\_rq : 2 \mid (FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx)\ FlgRq(Cs2.Ag, Fr.To.Dt.Tm.Mx))$
$(FlSt, \langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy\rangle)\ ,- ,$
$(AvSt(FG) \geq 2) \wedge Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx) \wedge (Fr = \texttt{"Las Vigas"}) \wedge (\texttt{"Jan."} = Dt) \wedge (Py := 2 * Cx * .70)$
$\langle Tflg\_cl : 1 \mid - ,- ,(Cx \in Fm) \wedge$
$(\texttt{"15 Dec."} \leq Dt \leq \texttt{"30 Dec."}) \wedge (Rfnd := Vat(Py)) \vee (\texttt{"May"} \leq Dt \leq \texttt{"Aug."} \wedge (Rfnd := Py * .5))$
. . . .
$\langle Tflg\_cl : vc \mid (FlCl, IT_1^c)\ (FlSt, IT_2^c)\ , (FlRfd, OT_1^r)\ (Flcld, OT_2^r)\ , CD^c\rangle$
$\langle Tflg\_bk : vb \mid (FlBk, IT_1^b)\ (FlSt, IT_2^b)\ , (FlBkd, OT_1^r)\ (FlPay, OT_2^r)\ , CD^b\rangle$
$\langle Tflg\_rq : vr \mid (FlRq, IT_3^r)\ (ChkS, IT_1^r)\ (FlSt, IT_3^r)\ , (FlRs, OT_1^r)\ , CD^r\rangle$
The Flight Service Interface Behavioural Specification
Imported (received) messages
Exported (invoked) messages
ChkSeat
ChSt(...)
FlSt
$\langle Fg1|FgInf : [K89.Uml.Paris.12306.1430.230], AvSt(Fg)..\rangle$
$\langle Fg2|FgInf : [I24.London.DC.02606.2245.817], AvSt(Fg)..\rangle$
$RsSt(Cs, Fg) \bowtie IT_1^r$
$FlgRq(Cs.Ag, Fr.To.Dt.Tm.Mx) \bowtie IT_2^r$
$\langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy\rangle \bowtie IT_3^r$
Tflg_rq
$(AvSt(FG) \geq 1) \wedge Rs.[Cs.R] \wedge (Cx \leq Mx) \wedge (Py := Cx)$
$\wedge((Ag \leq 18) \wedge (Py := Cx * 08))) \bowtie CD^r$
$FlgRqd(Cs, Fg, [R.Fr.To.Dt.Tm], Py, Dy) \bowtie OT^r$
FlRq
FlgRq(Ann,..)
FlRs
FlgRsv(fl1,..)
FlBkd
Flgbkd(Cs,..)
$FlgBk(Cs, R, Dy, Py) \bowtie IT_1^b$
$\langle FG|FgInf : [R.Fr.To.Dt.Tm.Cx], Rsv.Rs, Cmf : Fm\rangle \bowtie IT_2^b$
Tflg_bk
$(Dc \leq Dy) \wedge (Cs \in Rs) \wedge (Py = Cx) \wedge (Pn = 0)\wedge$
$(Cmf.[Cs.R]) \wedge ((Dc \geq Dy) \wedge (Pn := Py * 0.1)) \bowtie CD^b$
$FlgBkd(Cs, R.Fr.To.Dt.Tm, Py) \bowtie OT_1^b$
$FlgPay(Cs, R, Py) \wedge FlgPnl(Cs, R, Pn) \bowtie OT_2^b$
FlBk
FlgBk(...)
FlPay
FlgPay(Cs,...)
PlgPnt(Cs,...)
$FlgCl(Cs, R, Py, Dt) \bowtie IT_1^c$
$\langle Fg|FgInf : [R.Dt], Rsv : Rs, Cmf : Fm\rangle \bowtie IT_2^c$
Tflg_cl
$((Cs \in Rs) \wedge (Rfnd := Py)) \vee ((Cs \in Fm) \wedge (Dc \leq Dt)\wedge$
$(Rfnd = Py * 0.85)) \vee ((Cs \in Fm) \wedge (Rfnd = Py * 0.45)) \bowtie CD^b$
$FlgRfnd(Cs, R, Rfnd) \bowtie OT_1^c$
$FlgCld(Cs, R) \bowtie OT_2^c$
FlCl
FlgCl(anni,...)
FlRfd
FlgRfnd(Cs,..)
Flcld
FlgCl(Cs,...)

Figure 6.3: The Runtime Adaptability of CSrv-Nets flight service

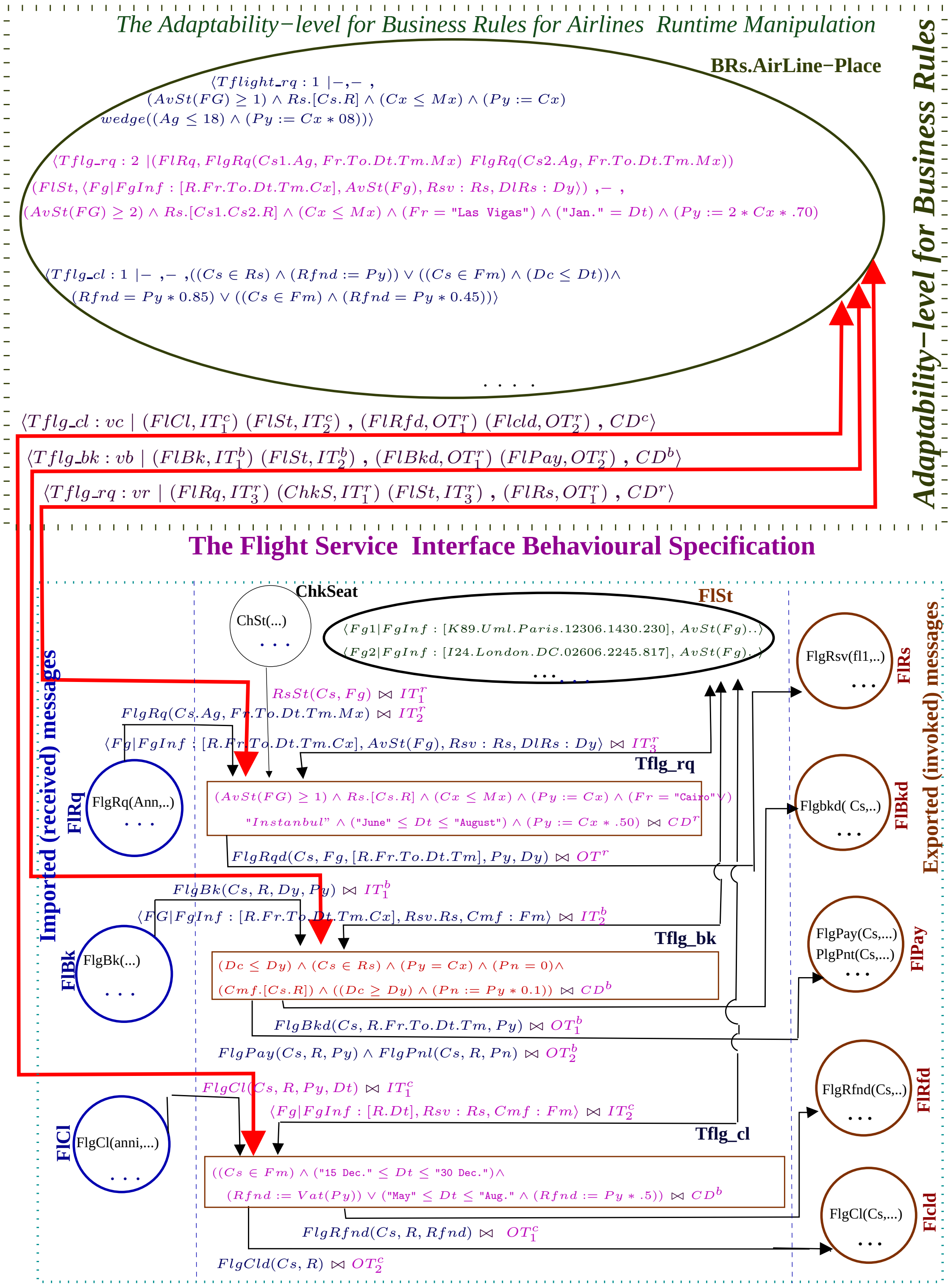

The Adaptability−level for Business Rules for Airlines Runtime Manipulation
BRs.AirLine−Place
Adaptability−level for Business Rules
The Flight Service Interface Behavioural Specification
ChkSeat
FlSt
ChSt(...)
FlgRq(Ann,..)
FlgBk(...)
FlgCl(anni,...)
FlgRsv(fl1,..)
Flgbkd(Cs,..)
FlgPay(Cs,...)
PlgPnt(Cs,...)
FlgRfnd(Cs,..)
FlgCl(Cs,...)
FlRq
FlBk
FlCl
FlRs
FlBkd
FlPay
FlRfd
Flcld
Tflg_rq
Tflg_bk
Tflg_cl
Imported (received) messages
Exported (invoked) messages

Figure 6.4: (Shifted) Runtime Adaptability of CSrv-Nets flight service

Chapter 7

Conclusions and Future Work

The main objective that we have been focussing on during this thesis concerns the formal specification and validation of dynamically adaptive knowledge-intensive service-driven business applications. The thesis puts forwards an integrated and progressive approach based on high-level Petri nets and event-driven business rules. In this closing chapter, we first summarise the main achieved contributions. Secondly, we present some of the possible extensions of this work towards a more complete approach and methodology.

7.1 Thesis Contribution

As the service-oriented paradigm with its Web services technology and standards are getting increasingly maturing, more and more (worldwide) cross-organisations and governmental institutions are shifting towards this technology at a fast pace. Among other critical requirements, this service-oriented tendency implies complex composite service applications to be engineered and developed, that should exhibit at least the following features. Firstly, most potential service applications such as E-Commerce, E-health and E-Government are by essence *knowledge-intensive*, with more specifically event-driven business rule-centric applications. Secondly, these service-driven applications are mostly mission critical, which implies that only a high degree of preciseness and *formality* are able to ensure this feature. Thirdly, to cope with swift competition, market volatility and economy changes, these service-driven applications imperatively are required to be dynamic, evolving and adaptive.

In this thesis, we have thus been addressing these three crucial features in today's complex service-driven applications, namely: business rule-centricity; formal foundation and runtime adaptability. More precisely, we put forward an integrated and progressive approach for specifying and validating service-driven rule-centric and adaptive business applications. The approach is based on an innovative variant of high-level Petri nets, we referred to as adaptive CCSrv-Nets. This framework can be distinguished at least with the following characteristics:

Stateful and Conversational: The fact that CCSrv-Nets intrinsically permits coping with both the *type* (e.g. as algebraic service specification) and the service *instances* (i.e. service states and message instances), we demonstrated how they are able to deal with persistency. Conversation is dealt with the using of any partial-ordering of transitions (as business-activities), that is, parallelism, sequence, choices and so on.

Inherently Rule-centric: CCSrv-Nets transitions are incrementally built to directly reflect the corresponding event-driven business rule. In particular, any conditions that are complex are straightforwardly constructed using first-order expressions with the involved message parameters and properties values (as variables).

Validation through Visualisation: As we demonstrated any CCSrv-Nets Web services specification can be validated by animating it with any initial configuration (ie, set of service states and message instances). The validation purposes includes the detection of errors, misconceptions and failure. The fact that the validation is graphical and can be achieved with simultaneous firing of different activities make it very attractive and understandable.

Locally and Globally Compositional: As we developed respectively in chapter 4 and 5, we proposed two levels for specifying and validated rule-centric service-driven applications. First, the usual so-called service orchestration can be achieved by focussing on one service and calling others; for instance we specified Airline services which may call the banking services. The second level of specification concerns the so-called choreography where different services have to be invoked and composed with the same priorities; the example we developed concerns the Agency service which integrates different services including Airlines, Hotels, and Banking.

Behaviourally Runtime Adaptable: The most significant challenging contribution of this thesis concerns the dynamic adaptability, where we are requested to be highly innovative. We thus put forward on top of each CCSrv-Nets specification an extra adaptability-level and linked it to the base-level in a way that business rules can be dynamically shifted up and down.

In summary the most significant contributions of this thesis with respect to the formal engineering of adaptive and knowledge-intensive service-driven business applications could be highlighted in the following points:

A formalism for Specifying and Validating these Applications: The formalism is progressively constructed by first semi-formally modelling of the concerned service-driven applications using UML class-diagrams and business rules. The CCSrv-Nets formalism as we just emphasised permits to intrinsically integrate such business rules in the modelled service-oriented business process, with the possibilities of dealing with both orchestration and choreography in a harmonious complementary manner.

Inherent Design-time Rule-centric Adaptability: Given the fact that we have been intrinsically integrating event-driven business rules, the resulting CCSrv-Nets specification is by construction very flexible and adaptable. Moreover, at the design-time any modelled business rules as transitional behaviour can be updated as requested. More precisely, we can change the conditions and/or any input/output arc-inscriptions of the chosen transition.

Emerging Runtime Behavioural Adaptability: As we emphasised this is the most innovative contribution of this thesis, so far no proposals have been able to cope with runtime adaptability as we put forward in this thesis. We were thus able to dynamically weave/unweave any business rule as tuple from/to the adaptability-level, where rules can be dynamically manipulated (i.e. added, removed or changed).

7.2 Future Work

After having put forward this crucial step towards rigorously modelling and validating adaptive and rule-centric service applications, we are aware that more milestones are required towards an integrated, practical and methodologically supported approach. More specifically, we are aware that much work remains ahead at least on the four research and practical directions.

7.2.1 Deployment Using Advanced Web Standards

In this thesis we have been exclusively concentrating on the foundational-level, through hinting on how the deployment-phase using advanced web services technology and standards can be achieved. We thus project that such deployment must be addressed in more detail in any future extension of the proposed approach. In particular, once a specification is corrected, validated and adapted it should be mapped to corresponding web standards such as WSDL and WS-BPEL. Nevertheless, since such standards are static and purely process-centric, to achieve a preserving and compliant mapping we require a more advanced enhancement of such standards. The integration of business rules within BPEL as achieved in [49] could be a promising starting point. Another interesting direction towards that aim is the adoption of aspect-oriented techniques as recently suggested in [19].

7.2.2 Supporting Tools For the Approach

After putting forwards the main milestones and concepts behind the CSrv-Nets formalism for orchestrating rule-centric service-driven applications, we found ourselves enforced to continue in one of the two following directions: (1)

Building an advanced supporting software tools for environment using the JAVA technology; or (2) fundamentally enhance this approach towards choregraphical composition (i.e. as developed in chapter 5) and come up with an extra adaptability-layer for coping with runtime adaptability (chapter 6). Indeed, both directions are very time-consuming and difficult to address. We have thus decided to continue in the fundamental direction, though being highly aware of the necessity of supporting tools for the approach.

Enhancing the practicability of this approach requires the development of supporting software tools. These tools should include at least: (1) an editor simulator for CCSrv-Nets specification, that is, this tool should allow the designer to describe the specification, correct it and validate it using graphical simulation as we highlighted in the thesis. The second complementing tool should cope with the runtime adaptability, that is, it should permit the manipulation of business rules as tuples, dynamically bringing them down to the base-level and animating them as we showed in detail in the previous chapter.

7.2.3 Extensions Towards Formal Verification

Since our research group has wide experience in the formal verification of complex systems using the temporal setting, it would be a promising direction to extend this work towards verification, instead of just modelling and validation. More specifically, we argue that the research explored by [89] in leveraging ITL towards composing and verifying behavioural Web services could be a good starting point in that direction. Indeed, we claim that it is quite possible since our transitions could be easily expressed in this ITL-Web services framework.

References

[1] Kavantzas, N and Olsson, G and Mischkinsky, J. Web Services Choreography Description Language 1.0. http://www.w3.org/TR/ ws-cdl-10/, w3.org, 2004.

[2] Aalst, W. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[3] Aalst, W. Don‘t Go with the Flow: Web services Composition Standards Exposed. *IEEE Intelligent Systems*, 18:72–76, 2003.

[4] Andrew, S and Tanenbaum, M. *Distributed Systems: Principles and Paradigms.* Prentice Hall PTR, 2001.

[5] Ankolekar, A. DAML-S:Web Service Description for the Semantic Web. In *Proc. of International Semantic Web Conference (ISWC)*, pages 348–363. IEEE CS, 2002.

[6] Apt, K. Ten Years of Hoare's Logic : A Survey - Part I. *TOPLAS*, 3(4):431–483, 1981.

[7] Asir, V. *Web Services Description Language(1.1).* W3C Recommendation, http://www.w3.org/TR/wsdl, 2001.

[8] Baral, C and Gelfond, M. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[9] Barkaoui, K. Dutheillet, C and Haddad, S. An Efficient Algorithm for Finding Structural Deadlocks in Colored Petri Nets. In *Proc. of 14th Int. Conf. on Application and Theory of Petri nets*, Lecture Notes in Computer Science. Springer, 1993.

[10] Bernard, B and Michel, D. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans, Software Eng*, 259-273, 1991.

[11] Berners, L. Weaving the Web, 2000.

[12] Berners, L and Ora, L. *The Semantic Web.* Scientific Amrican, May 2001.

[13] Biggs, N and Wilson, R. *Graph Theory.* Oxford University Press, 1986.

[14] BM, C. *Web Services Transaction.* www.ibm.com/developerworks/web/library/ws-transpec/, 2000.

[15] Brauer, W and Reisig, W and Rozenberg, G. Petri Nets: Central Models and Their Properties. In *Proc. of 18th Int. Conf. on Application and Theor of Petri nets*, volume 254, 1986-1987.

[16] Cau, A and Moszkowski, B and Zedan, H. Interval Temporal Logic, 2005.

[17] Cazzola, W and Chiba, S and Ledoux, T. Reflection and Meta-Level Architectures: State of the Art, and Future Trends. In *ECOOP'2000 Workshop Reader*, volume 1964 of *lncs*, pages 1–15. Springer, 2000.

[18] Cazzola, W and Stroud, R and Tisato, F, editor. *Reflection and Software Engineering*. Lecture Notes in Computer Science Vol. 1826, Springer, 1996.

[19] Charfi, A and Mezini, M. Hybrid Web Service Composition: Business Processes Meet Business Rules. In *Proceedings 2nd International Conference on Service Oriented Computing (ICSOC04)*. ACM Press, 2004.

[20] David, B. Web Services Architecture, W3C Working Group Note, 2004.

[21] Dey, A. Towards a Services Platform for Context-Aware Applications, 2003.

[22] Duan, Z and Bernstein, A and Lewis, P. A Model for Abstract Process Specification, Verification and Composition. In *Proceedings of the 2nd international conference on Service oriented computing (ICSOC'04)*, pages 232–241. ACM Press, 2004.

[23] Edward, J. The Rule Markup Initiative. *www.ruleml.org*, 2005.

[24] Ehrig, H and Mahr, B. Fundamentals of Algebraic Specifications 1 : Equations and Initial semantics. *EATCS Monographs on Theoretical Computer Science*, 21, 1985.

[25] Ehrig, H and Padberg, J. Graph Grammars and Petri Net Transformations. *Lecture Notes in Computer Science*, 3098:496–536, 2004.

[26] Elrad, T and Filmanand, R and Bader, A. Special Issue on Aspect Oriented Programming. *Communications of the ACM*, 44(10), 2001.

[27] Eric, N. *Understanding Web Services*. Addision-Wesley, September 2002.

[28] Foster, H. Behaviour Analysis And Verification of Web Service Composition, note = Phd Thesis, University of london, 2004.

[29] Foster, H. A Rigorous Approach To Engineering Web Service Compositions, 2006.

[30] Goguen, J and Diaconescu, R. An Oxford Order Sorted Algebra. *Mathematical Structures in Computer Science*, 4(4):363–392, 1994.

[31] Goguen, J and Winkler, T and Meseguer, J. Introducing OBJ. Technical Report SRI-CSL-92-03, Computer Science Laboratory, SRI International, 1992.

[32] Grady, B and James, R and Ivar, J. *Unified Modeling Language, Notation Guide, Version 1.0*. Addison-Wesley, 1998.

[33] Haas, H. Web Services Activity. Technical report, W3C Web Services Activity Group, 2002.

[34] Hamadi, R and Benatallah, B. A Petri Net-based Model for Web Service Composition. In *Proceedings of the 14th Australasian Database Conference*, volume 17 of *CRPIT*, pages 191–200. Australian Computer Society, 2003.

[35] Hamadi, R and Benatallah, B and Medjahed, B. Self-adapting Recovery Nets for Policy-driven Exception Handling in Business Processes. *Distrib Parallel Databases*, 23:1–44, 2008.

[36] Heather, K. Web services Conceptual Architecture, WSCA 1.0. Technical report, IBM Software Group, *http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf*, 2001.

[37] Heckel, R and Mariani, L. Automatic Conformance Testing of Web Services. In *Proceedings FASE*, volume 3442, pages 34–48, 2005.

[38] Hewlett, P. *Web Services Conversation Language (WSCL)1.0.* W3C Recommendation, http://www.w3.org/TR/wscl10/, 2002.

[39] Hunter, D, editor. *Beginning XML.* Wiley Publishing, 2004.

[40] ISO, 1995. Open Distributed Processing - Reference Model - Part2: Foundations, International Standard 10746-2 / ITU-Recommendation X.902.

[41] James, B. Alexandra, P and Peter, W. An Event-Condition-Action Language for XML. In *Birkbeck College*. University of London, 2003.

[42] Jensen, K. Coloured Petri Nets: Basic Concepts, Analysis Methods and practical Use - Volume 1 : Basic Concepts. *EATCS Monographs in Computer Science*, 26, 1992.

[43] Jensen, K and Rozenberg, G. *High-level Petri Nets.* Springer, 1991.

[44] Jhon, E. *Web Services Coordination.* www.ibm.com/developerworks/web/library/ws-coor, 2000.

[45] John, D and Holger, L. Web Service Modelling Ontology. Technical report, *http://www.wsmo.org*, 2004.

[46] Joseph, W. The Web services Debate: J2ee vs Net. pages 58–63. Commun, ACM, 2003.

[47] K, Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, volume 1 of *Basic Concepts. Monographs in Theoretical Computer Science.* Springer, 1979.

[48] Kadir, W and Loucopoulos, P. Relating Evolving Business Rules to Software Design. *Journal of Systems Architecture*, 2003.

[49] Kardasis, P and Loucopoulos, P. Expressing and Organising Business Rules. *Information and Software Technology*, 2004.

[50] Kiczales, G. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. LNCS, 1997.

[51] Kiczales, G. An Overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'01)*, pages 327–353. LNCS 2072, 2001.

[52] Knolmayer, G and Endl, R and Pfahner, M. Modeling Processes and Workflows by Business Rules. In *Business Process Management*, volume 1806, pages 16–29. Springer, 2000.

[53] Lakos, C. From Coloured Petri Nets to Object Petri nets. In *Proc. of 16th Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 278–287. Springer, 1995.

[54] Lakos, C. The Consistent Use of Names and Polymorphism in the Definition of Objects Petri Nets. In *Proc. of 17th Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science*, pages 380–399. Springer, 1996.

[55] Lang, P and Obermair, W and Schrefl, M. Modeling Business Rules with Situation / Activation Diagrams. In *Proc. of the 13th International Conference on Data Engineering (ICDE)*, pages 455–464. IEEE Computer Society Press, 1997.

[56] Leymann, F. Web Services Flow Language (WSFL 1.0). Technical report, IBM Academy Of Technology, 2001.

[57] Martens, A. On Compatibility of Web Services. In *Petri Net Newsletter, Volume 65*, pages 12–20, 2003.

[58] Martens, A. On Usability of Web Services. In *Proceedings of 1st Web Services Quality Workshop (WQW 2003)*, 2003.

[59] Martens, A. Analyzing Web Service Based Business Processes. In *Proceedings FASE 2005*, volume 3442, pages 19–33, 2005.

[60] Martin, D, Paolucci, M and Mcilraith, S, editor. *Bringing Semantics to Web Services: The OWL-S Approach*, volume 3387 of *Lecture Notes in Computer Science*. Springer, 2004.

[61] Martin, G and Marc, H. *SOAP Version 1.2 Part 1: Messaging Framework.* W3C Recommendation, World wide Web Consortium, June 2003.

[62] Massimo, A and Walter, C. Implementing the Essence of Reflection: a Reflective Run-Time Environment. In *Proceedings of the 9th Annual ACM Symposium on Applied Computing (SAC'04)*, Nicosia, Cyprus, 2004. ACM Press.

[63] Masuhara, H and Yonezawa, A. A Reflective Approach to Support Software Evolution. In *Proceedings of International Workshop on the Principles of Software Evolution*, pages 135–139, 1998.

[64] Mecella, M and Presicce, F and Pernici, B. Modeling E-service Orchestration Through Petri Nets. In *TES 2002*, volume 2444, pages 38–47, 2002.

[65] Meredith, G and Bjorg, S. Contracts and Types. *Communications of the ACM*, 46(10):41–47, 2003.

[66] Meseguer, J. A Logical Theory of Concurrent Objects and its Realization in the Maude Language. In *Research Directions in Object-Based Concurrency*, pages 314–390. The MIT Press, 1993.

[67] Microsoft Corporation. The Component Object Model Specification, 1995.

[68] Microsoft, G. Microsofts NET Homepage. Technical report, 2005.

[69] Moldt, D and Offermann, S and Ortmann, J. A Proposal for Petri Net Based Web Service Application Modeling. In *In Proceedings of ICWE 2004*, volume 4140, pages 93–97. Springer, 2004.

[70] Narayanan, S and McIlraith, S. Analysis and Simulation of Web Services. *Computer Networks*, 42:675693, 2003.

[71] Object Management Group. The Common Object Request Broker: Architecture and Specification(corba)rev 3.0.2.technical document. Technical report, OMG, 2004.

[72] Orrins, B and Yang, J and Papazoglou, M. A Framework for Business Rule Driven Web Service Composition. In *Proc. of Conceptual Modeling for Novel Application Domains*, volume 2814, pages 52–64. Springer, 2003.

[73] Paananen, J. Introduction to and Comparision of Formalisms, 1995. *www.tml.hut.fi.*

[74] Peterson, J. Petri Net Theory and the Modeling of Systems. In *Englewood Cliffs*, Inc. New Jersey: Prentice Hall, 1981.

[75] Petri, C. Kommunikation mit Automaten, 1962.

[76] Piotr, C and Boualem, B and Rachid, H. A Top-Down Petri Net-Based Approach for Dynamic Workflow Modeling. *of Lecture Notes in Computer Science*, pages 336–353, jan 2003.

[77] Poladian, V and Pedro, S and Shaw, M. Dynamic Configuration of Resource-Aware Services. In *Proceedings of the 26th International Conference on Software Engineering (ICSE).* ACM Press, 2004.

[78] Reisig, W. Petri Nets. *Springer-Verlag EATCS Monographs on Theoretical Computer Science*, 4, 1982.

[79] Reisig, W. Petri Nets and Abstract Data Types. *Theoretical Computer Science*, 80:1–30, 1991.

[80] Reisig, W. Petri Nets and Algebraic Specifications. *Theoretical Computer Science*, 80:1-24, 1991.

[81] Richters, M and Gogolla, M. Validating UML Models and OCL Constraints. In *Proc. 3rd Int. Conf. Unified Modeling Language (UML'2000)*, LNCS. Springer, 2000.

[82] Robert, B. *Web services and Flows (WSFL).* Sams Publishing, September 2002.

[83] Roger, W. XML Web Services Basics. Technical report, Microsoft Corporation, June 2002.

[84] Rosca, D and Wild, C. Towards a Flexible Deployment of Business Rules. *Expert Systems with Applications*, 23:385–394, 2002.

[85] Schmidt, K. Verification of Siphons and Traps for algebraic Petri nets. In *Proc. of 18th Int. Conf. on Application and Theory of Petri nets*, Lecture Notes in Computer Science, pages 427–446. Springer, 1997.

[86] Sibertin, C. Communicative and Cooperative Nets. In *Proc. of the 15th International Confernce on the application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*. Springer, 1994.

[87] Sid, A and Ben, B. Web Service Business Process Execution Language Version 2.0. Technical report, OASIS, http://www.oasis-open.org/apps/org/workgroup/wsbpel/, December 2004.

[88] Skogan, D and Groenmo, R. Web Service Composition in UML. *IEEE International*, pages 47–57, Sept 2004.

[89] Solanki, M. A Compositional Framework for the Specification, Verification and Runtime Validation of Reactive Web services. *Phd Thesis*, 2005.

[90] Solanki, M and Cau, A and Zedan, H. Introducing Compositionality in Web Service Descriptions. In *Proceedings of the International Conference on World Wide Web*. IEEE Computer Society Press, 2004.

[91] Tabet, S and Wagner, G and Boley, H. MOF-RuleUML: The Abstract Syntax of RuleML as a MOF Model. In *Integrate*, 2003.

[92] Technical. UDDI Technical White Paper. Technical report, OSSIS, http://uddi.org/pubs/uddi-tech-wp.pdf, October 2004.

[93] Thatte, S. XLANG Web Services For Business Process Design, 2001.

[94] Valk, R. Concurrency in Communicating Object Petri Nets. In *Concurrent Object Oriented Petri Nets*, volume 2001, pages 164–165. Springer, 2001.

[95] Van, A and Basten, T. Life-Cycle Inheritance: A Petri-Net-Based Approach. In *Proc. of ICATPN*, pages 62–81, 1997.

[96] W3C. XML Schema. Technical report, http://www.w3c.org/XML/Schema, 2004.

[97] Wermlinger, M and Fiadeiro, J. A Graph Transformation Approach to Software Architecture Reconfiguration. *Science of Computer Programming*, 44:133–155, 2002.

[98] Winskel, G and Nielsen, M. Models for Concurrency. *Handbook of Logic in Computer Science*, 4, 1992.

[99] Yi, X and Kochut, K. A CP-nets-based Design and Verification Framework for Web Services Composition. In *IEEE International Conference on Web Services, San Diego , California*, pages 756–760, 2004.

[100] Yi, X and Kochut, K. A CPNets-based Framework for Design and Analysis for Service Oriented Distributed Systems. *ACM Transaction on Internet Technology*, 2005.

[101] Yonezawa, A and Matsuoka, S. *Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192. Springer, Proc. Reflection, 2001.