

# On the Stepwise and Disciplined Engineering of Adaptive Service-Oriented Applications

H a b i l i t a t i o n s s c h r i f t

Zur Erlangung der Venia legendi für Informatik,

angenommen durch die Fakultät für Informatik  
der Otto-von-Guericke-Universität Magdeburg

Von Dr. Ing. Nasreddine Aoumeur

geb. am 16 März 1964 in Mascara

Gutachter

Prof. Gunter Saake (Otto-von-Guericke-Universität Magdeburg)

Prof. Djamel Benslimane (Claude Bernard University, Lyon, France)

Prof. Karsten Wolf (Universität Rostock)

Magdeburg, den 6.10.2010

# Acknowledgment

I am very grateful to Prof. Gunter Saake for offering me a scientific home all along this years. The fruitful discussions within the Database group allowed me to significantly reshape and improve the output of this work. Therefore, many thanks to each member of the group. I am also thankful to the dean of our faculty, for providing me with all requested resources to achieve this work in best conditions. The presented research results have grow out from my postdoctoral stay at the University of Leicester, within the European Agile project. Therefore, I express my gratitude to all members of the Project, and particularly Profs. Jose Fiadeiro and Martin Wirsing for their constructive conversations and advices. I am specially further grateful to Prof. Kamel Barkaoui, for the long-term and promising cooperation and the continuous support and encouragement. My particular thanks go also to the reviewers of this work. For the practical side of this thesis, I am particularly thankful to Msc. Ur Rahman Saif for implementing the approach vision on aspectual .NET and Msc. Liu Manru for implementing the extended aspect-oriented MAUDE. I am forever indebt to my fathers for supporting me on all stations of my life; your caring and love will always enlighten my path. My big love goes also to my small family and kids. Last but not least, I am graceful to my God for empowering me with all the strengths and faith to achieve this modest deed.

# Abstract

Service technology geared by its SOA architecture and enabling Web-Services is rapidly gaining in maturity and acceptance. Consequently, most of world-wide (private and corporate) cross-organizations are embracing this paradigm by publishing, requesting and composing their businesses and inherent applications in 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 Services standards such as WSDL, BPEL, WS-CDL and many others offer just static, manual and purely process-centric knowledge-scarce ad-hoc techniques to deploy such services. Furthermore, current research proposals to leverage such standards towards more correctness and adaptability are still in their infancy stages and do not thus scale up to realistic and wide adoption. Indeed, potential service-oriented applications such as E-commerce, E-Banking and E-health are required to be highly adaptive and dependable, while being mostly governed by volatile *rule-centric* knowledge.

The main aim of this thesis consists therefore in leveraging the development of service-oriented applications towards more reliability, dynamic adaptability and knowledge-intensiveness. After a throughout study and critical analysis of the current state-of-art, this thesis puts forwards an innovative stepwise and disciplined approach towards engineering and deploying dynamically adaptive rule-centric service-oriented applications. More specifically, the approach starts by intuitively eliciting structural service features through stereotyped service-based UML-class diagrams. For the behavioral service features, the approach proposes to govern any involved business activity through respective intensional event-driven business rules, we then leverage towards operational architectural ECA-driven rules. For the crucial conceptual phase, the approach puts forwards a tailored service-oriented Petri nets framework, we refer to as adaptive CSRV-NETS, that exhibits the following potential characteristics. First, the framework smoothly builds on the previous business-level phase, by soundly integrating behavioral event-driven business rules and stateful services, both at the type and instance level. Second, with its intrinsic true-concurrent semantics based on rewriting-logic, the framework provides formal validation through a tailored and compliant extension of the MAUDE language and its reflection capabilities. Third, the framework explicitly separates between orchestration for modelling rule-intensive single services and choreography for cooperating several services through their balanced governing interactive business rules. Fourth, by capitalizing on aspect-oriented potentials for separation of concerns and adaptability, the framework is smoothly shifted towards runtime adaptability, through a compliant aspectual-level. Such adaptability-level allows for dynamically shifting up and down any rule-centric behavior of the running CSRV-NETS-based service-components. Last but not least, towards bridging the gap to Web-Service technology, we developed an aspectual .Net framework that is fully compliant with the above approach founded phases.

# Zusammenfassung

Serviceorientierung mit ihrer Serviceorientierten Architektur (SOA) und die darauf basierende praxisorientierte Umsetzung in Form von Webservices gewinnen heutzutage zunehmend an Reife und Bedeutung. Folglich verwenden viele weltweit operierende Unternehmen und Organisationen dieses Paradigma, um ihre internen wie auch unternehmensübergreifenden Geschäftsprozesse umzusetzen. Insbesondere werden diese Geschäftsanwendungen und Prozesse in Form von Webservices konzipiert, veröffentlicht, angefordert und komponiert. Der stärkere wirtschaftliche Wettbewerb erfordert, dass diese unternehmensübergreifenden Dienstanwendungen anpassbarer, wissensbasiert und hoch zuverlässig sein müssen. In Gegensatz zu diesen Forderungen bieten gegenwärtige Webstandards wie WSDL, BPEL oder WS-CDL lediglich statische, rein prozesszentrische und wenigfundierte Techniken. Forschungsvorhaben zur Verbesserung dieses Stands hinsichtlich mehr Korrektheit und Anpassungsfähigkeit sind noch zu unausgereift, um realistische Szenarien zu bewältigen. Gleichzeitig verlangen potenzielle Einsatzgebiete dienstorientierter Anwendungen wie elektronischen Handel, Online-Banking und elektronisches Gesundheitswesen nach hoher Anpassungsfähigkeit und Zuverlässigkeit. Ferner sind diese Anwendungen von sich schnell änderndem, regelbasiertem Verhalten gesteuert.

Diese Arbeit setzt sich zum Hauptziel, die Entwicklung von serviceorientierten Anwendungen für mehr Zuverlässigkeit, dynamische Anpassungsfähigkeit und Unterstützung von Geschäftsregeln voranzutreiben. Nach einer umfassenden Studie und kritischen Analyse des gegenwärtigen Stands der Technik, führt diese Arbeit einen fundierten Ansatz für die Entwicklung von dynamischen, anpassungsfähigen serviceorientierten Anwendungen ein. Der Ansatz beginnt mit einer intuitiven Beschreibung der strukturellen Eigenschaften von Services durch stereotypisierte, dienstbasierte UML-Klassendiagramme. Für die Beschreibung des Verhaltens von Services schlägt der Ansatz für jede betroffene Geschäftsaktivität adaptive intentionale ereignisgesteuerte Geschäftsregeln vor. Anschließend betrachten wir die Umsetzung dieser intentionalen Geschäftsregeln in eine ECA-gesteuerte Architektur. Für die entscheidende formale Phase schlägt der Ansatz einen maßgeschneiderten serviceorientierten Petri-Netz-Formalismus vor. Dieses als CSRV-NETS bezeichnete konzeptuelle Modell verfügt über folgende Eigenschaften: Das vorgeschlagene Framework baut auf der vorhergehenden intuitiven Phase auf, indem es ECA-Geschäftsregeln und zustandsbehaftete Dienste auf Typ- als auch Instanzebene integriert. Zweitens ermöglicht das Framework eine formale Validierung, einerseits mit Hilfe seiner nebenläufigen, auf Termersetzung basierten Logik, und andererseits durch eine maßgeschneiderte Erweiterung der Sprache MAUDE. Drittens, das konzeptuelle Modell trennt explizit zwischen der Orchestration für die Modellierung von Einzelservices und der Choreografie von globalen, kooperierenden Diensten durch balancierte dienstübergreifende Geschäftsregeln. Viertens, die Verwendung von aspektorientierten Techniken zur Extraktion von querschneidenden Belangen und die daraus resultierende Anpassbarkeit, ermöglicht uns eine trans-

parente Erweiterung des CSRV-NETS in Richtung regelbasierter Laufzeitanpassungen. Diese Adaptationsebene erlaubt unter anderen das Einweben von regelbasierten Verhalten in laufende CSRV-NETS-Komponenten. Um schließlich die Lücke zu Webservice-Techniken zu überwinden, haben wir ein aspektorientiertes .Net-Werkzeug entwickelt, welches den entwickelten fundierten Ansatz vollständig umsetzt.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation and work scope . . . . .	1
1.2	Main envisioned work's results . . . . .	4
1.3	Work Outline . . . . .	5
<b>2</b>	<b>Web-Services Foundation and Adaptability: Survey and Criteria</b>	<b>8</b>
2.1	SOA and Web-Services: Overview and main Ingredients . . . . .	9
2.1.1	The Underlying Technologies for SOA . . . . .	10
2.1.2	Services-Oriented Architecture (SOA) . . . . .	13
2.1.3	Web services Specification and Composition Standards . . . . .	14
2.2	(High-Level) Petri nets-Based Foundation for WS: Survey . . . . .	17
2.2.1	P/T Nets-based Foundations for Web services . . . . .	18
2.2.2	Modelling Web services with High-level Petri Nets . . . . .	21
2.3	Service Adaptability: Rules- and Aspect-based proposals . . . . .	24
2.3.1	Business Rule-driven Proposals to Web-Service Adaptability . . . . .	25
2.3.2	AOP and Adaptive Service-oriented Applications . . . . .	27
2.4	Web-Services Modelling and Adaptability: Criteria and Assessment . . . . .	28
2.4.1	Criteria for Web-Services Modelling and Adaptability . . . . .	29
2.4.2	Service composition criteria . . . . .	30
2.4.3	Service Criteria applied on the state-of-art . . . . .	33
2.5	Chapter Summary . . . . .	34
<b>3</b>	<b>Rule-centric Stepwise Development for Service Systems</b>	<b>35</b>
3.1	Rational for the forwarded Conceptual framework . . . . .	35
3.1.1	HLPN as service foundation: Potentials and limitations . . . . .	36
3.1.2	Necessity for Stepwise supporting Methodology . . . . .	38
3.2	The UML-ECA-based semi-formal services description . . . . .	40
3.2.1	Profiled UML class-diagrams: Application to the Travel Agency . . . . .	41
3.2.2	Stepwise ECA-driven Description for Service Behaviors . . . . .	42

3.3	CSRV-NETS: Structural Features Modelling . . . . .	51
3.3.1	Application to the Travel Agency . . . . .	54
3.4	CSRV-NETS: Behavioral Modelling of Services . . . . .	57
3.4.1	CSRV-NETS behavior from ECA-driven architectural rules . . . . .	59
3.5	CSRV-NETS: A Rewriting-logic based behavioral semantics . . . . .	62
3.5.1	An intuitive CSRV-NETS behavioral semantics . . . . .	62
3.5.2	CSRV-NETS Rewriting-logic based semantics . . . . .	63
3.6	CSRV-NETS behavioral validation: A tailored MAUDE extension . . . . .	68
3.7	Chapter Summary . . . . .	71
<b>4</b>	<b>Collaborative Services—Choreography meets Orchestration</b>	<b>73</b>
4.1	Choreographical Services Composition with CSRV-NETS: Further Motivations . . . .	75
4.1.1	Choreographical composition within the Travel-agency . . . . .	78
4.2	Business-Rules pattern for Behavioral Choreography . . . . .	78
4.2.1	Cross-service business rules for the Agency application . . . . .	81
4.3	Leveraging CSRV-NETS to ECA-driven Behavioral Choreography . . . . .	85
4.3.1	Structural features in CCSRV-NETS . . . . .	85
4.3.2	Behaviorally composing services with CCSRV-NETS . . . . .	87
4.4	CCSRV-NETS-based Formalization of the composite Travel-Agency . . . . .	90
4.5	Chapter Summary . . . . .	90
<b>5</b>	<b>From Design- to Runtime adaptive services—Foundation and Deployment</b>	<b>91</b>
5.1	CSRV-NETS Design-time service Adaptability: Potentials and Flaws . . . . .	93
5.2	CSRV-NETS-based Aspectual-level: Main Ideas and Concepts . . . . .	96
5.2.1	CSRV-NETS-transitions: Towards an "aspect"- <i>representation</i> . . . . .	96
5.2.2	CSRV-NETS-based <i>aspectual</i> -Level: Informal presentation . . . . .	99
5.2.3	CSRV-NETS-based <i>aspectual</i> -Level: Formal setting . . . . .	101
5.3	CSRV-NETS meets its Aspectual Net: Jointpoints and pointcuts at concerns . . . .	103
5.3.1	CSRV-NETS and its smooth Endowing with Jointpoints . . . . .	103
5.3.2	Pointcuts for Connecting CSRV-NETS-Joinpoints to the Aspectual Net . . . .	105
5.3.3	AOCSRV-NETS: Aspect-oriented CSRV-NETS-extension Formalization . . . .	107
5.4	Runtime (un)weaving of advices in AOCSRV-NETS: Principles and Formalization . .	108
5.4.1	"Non-woven" Rewriting rules governing aspect-oriented transitions . . . . .	110
5.4.2	Dynamic-Weaving by Inferring "Non-woven" Rules . . . . .	111
5.5	Aspectual Leveraging for Adapting the CSRV-NETS Flight Service . . . . .	114
5.5.1	Leveraging the CSRV-NETS Flight towards adaptability . . . . .	114
5.5.2	Building and dynamically adapting the flight AOCSRV-NETS . . . . .	114
5.5.3	Emerging the rules-as-advices at the aspectual-level . . . . .	117

5.5.4	Runtime shifting down / up of rules-as-advice on the Flight CSRV-NETS service . . . . .	118
5.6	An Aspect-oriented MAUDE for Validating AOCSRV-NETS . . . . .	118
5.6.1	Aspect-orientation of architectural ECA-driven rules for Dynamic Adaptability	122
5.6.2	Towards an ECA-Compliant aspect-orientation of MAUDE . . . . .	123
5.6.3	Dynamic (un)weaving of aspectual MAUDE service-interactions . . . . .	126
5.7	Towards a compliant .NET environment WS-deploying of AOCSRV-NETS . . . . .	128
5.7.1	<b>Mapping and manipulation of Conceptual ECA in .NET</b> . . . . .	129
5.8	Chapter Summary . . . . .	130
<b>6</b>	<b>Conclusions and Future Work</b>	<b>132</b>
6.1	Main achieved contribution . . . . .	132
6.2	Envisioned further investigations . . . . .	134
	<b>Bibliography</b>	<b>136</b>
<b>A</b>	<b>Algebraic Specifications, (High-level) Petri Nets, Rewriting logic and MAUDE:</b>	
	<b>Overview</b>	<b>144</b>
A.1	Algebraic specification: an overview . . . . .	144
A.2	(High-level) Petri-Nets: Main Concepts . . . . .	146
A.2.1	Place/Transitions Petri nets . . . . .	147
A.2.2	High-level Petri nets (HLPN): An overview: . . . . .	148
A.2.3	Object-oriented Petri Nets: An overview . . . . .	150
A.3	Rewriting techniques . . . . .	151
A.4	Rewriting logic . . . . .	153
A.4.1	Rewriting Logic and its Theory . . . . .	154
A.4.2	The meaning of Rewriting Logic . . . . .	155
A.5	MAUDE and its Reflection : Overview . . . . .	155
A.6	MAUDE main Features . . . . .	156
A.6.1	MAUDE Functional Modules . . . . .	157
A.6.2	System and object-oriented Modules . . . . .	157
A.6.3	MAUDE Reflection and internal Strategies . . . . .	160
A.6.4	Internal Strategies . . . . .	162
A.6.5	MAUDE-Workstation : presentation . . . . .	163
<b>B</b>	<b>N.Aoumeur Publications Related to this Thesis</b>	<b>166</b>



# List of Figures

1.1	On the Disciplined and Stepwise Engineering of Adaptive and Complex Service-Oriented Systems . . . . .	5
2.1	Web services stack . . . . .	11
2.2	Web services architecture . . . . .	14
3.1	Disciplined Approach for Developing Adaptive Service-Oriented Systems . . . . .	37
3.2	The approach two first phases, <code>UmlRule--CSvr-Nets</code> , as chapter focus . . . . .	40
3.3	StereoTyped UML-Classes for Services Applied on Travel-Agency. . . . .	43
3.4	The generic ECA-driven architectural service interactions pattern . . . . .	49
3.5	ECA-driven rule for the Flight-Request Activity . . . . .	50
3.6	ECA-driven rule for the Flight-Request Activity . . . . .	51
3.7	SteroTyped UML-Classes for Services Applied on The Airline Service. . . . .	55
3.8	The CSRV-NETS-based Behavioural Specification of the Flight Service. . . . .	61
3.9	Generic pattern for CSRV-NETS-transitions . . . . .	66
3.10	The CSRV-NETS Flight implemented using the extended MAUDE . . . . .	70
3.11	Concrete CSRV-NETS-Flight service configuration scenarios . . . . .	71
3.12	The resulting CSRV-NETS-Flight configuration by running the previous one . . . . .	71
4.1	The Choreography with CSRV-NETS as third phase in the forwarded approach . . . . .	75
4.2	An Illustrative Complementarity of Orchestration and Choreography in the CSRV-NETS Approach . . . . .	79
4.3	The general pattern of cross-service (choreographical) ECA-driven business rules . . . . .	80
4.4	The transition pattern for collaborating services within CCSRV-NETS . . . . .	88
4.5	Behavioral Choreographical Specification of Travel Agency Service . . . . .	89
5.1	Dynamic Adaptability and its Deployment as fourth phase in the forwarded approach . . . . .	92
5.2	The generic CSRV-NETS-transitions behavior . . . . .	97
5.3	Part of the Flight Service CSRV-NETS behavioral Specification. . . . .	99
5.4	The CSRV-NETS-based Aspectual Net for ECA-driven Rules dynamism . . . . .	100
5.5	Leveraging (generic) CSRV-NETS-transitions with aspectual-variables as Joinpoints . . . . .	104

5.6	Generic transitions for the two-level Aspectual AOCSRV-NETS formalism . . . . .	106
5.7	The Runtime Adaptable AOCSRV-NETS flight service before rules weaving . . . . .	116
5.8	The Runtime Adaptable AOCSRV-NETS flight service after rules weaving . . . . .	119
5.9	From ECA-driven service interactions into a Compliant Aspect-oriented MAUDE extension for AOCSRV-NETS . . . . .	120
5.10	Dynamic weaving of Std- and Crd-withdraw interaction rules . . . . .	127
5.11	The IDE Environment and its main functionalities . . . . .	128
5.12	The principles of mapping ECA-centric rules to the Aspectual .Net Env. . . . .	130
5.13	Translating steps from ECA-conceptual to the compliant .NET env. . . . .	130
A.1	The dining philosopher problem as a P / T-net. . . . .	148
A.2	The dining philosopher problem as an algebraic Petri net . . . . .	150
A.3	Concurrent rewriting of bank accounts. . . . .	160
A.4	Strategies control the rules execution. . . . .	163
A.5	General view of MAUDE Workstation. . . . .	164
A.6	The result split panel of MAUDE Workstation. . . . .	165

# List of Tables

2.1 Service criteria applied on proposals for service foundation and adaptability . . . . 32



# Chapter 1

## Introduction

### 1.1 Motivation and work scope

Along the recent years, we have been witnessing an increasingly dominating market globalization, geared by highly unpredictable volatility and fierce competition. In parallel to that, the incredible advances and confluence of computation and wireless communications have been boosting the pervasiveness of the internet and the World Wide Web to be available anywhere, anytime and through any communication-aware devices and channels. As consequence, on the one hand, to stay competitive most of (private and public) organizations and institutions are being intensively collaborating their know-how. Thereby, they are *dynamically* building loosely-coupled networked cross-organizational giants. On the other hand, the internet has been leveraged from simple glue of (syntactical) information to an unavoidable complex scene for networking and composing such cross-organizational knowledge-intensive and interaction-centric realities.

Service-oriented computing (SOC) represents nowadays the best emerging technological innovations towards "faithfully" (semi-)automating these new business cross-organizational realities. Indeed, as a new computing paradigm, this technology treats distribution, loose-coupling and heterogeneity as the main 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, as the main enabling of service-oriented architectures (SOA), represent platform-independent self-contained software entities with explicit interfaces. Such interfaces are further adequate tailored to be universally described, published, discovered, composed and deployed on the Web. The readiness of Web-services to be (dynamically) composed from basic ones to form large scale evolvable business applications (e.g. C2B and B2B), represents undoubtedly the most distinguished feature of SOA—over other technology such as object and component-orientation.

As technology, Web-Services can thus be manipulated (e.g. described, published, discovered and composed) using adequate standards. These are described in terms of XML-based languages,

and mainly encompass: WSDL [Ved01] for service description, UDDI [Tec04] for service registry, SOAP for communication and BPEL4WS [AB04] and WS-CDL [ACKM04] for composing services (resp. as orchestration and choreography). These languages have been rapidly gaining in maturity and wide adoption. Consequently, most organizations are embracing this service technology, for automating their business and inherent networked information systems.

This significant technological shifting towards SOA and its enabling Web-Services at such rapid pace, has been pressing for more ad-hoc deployment service techniques [PTDL07]. In other words, we are going beyond ordinary process-centric static compositions of services, which traditionally adopt WS-BPEL as service-focussed orchestration and partly WS-CDL as global inter-service choreography. We are thus witnessing the emerging of "advanced" services that are mainly featured by the followings.

**Persistency and Conversation:** Complex realistic service-oriented business applications are mostly characterized as long-span live (e.g. E-Commerce, E-health, E-Banking). They are therefore required to be conversational and highly persistent. Handling persistency means providing advanced abstraction mechanisms, such as those provided by the object paradigm and its UML method [BJR98]), including classification, inheritance and roles. Unfortunately, even the widely adopted BPEL uses very restricted data-variables that vanish after execution.

**Knowledge-intensiveness:** Potential Service-oriented applications (e.g. E-commerce, E-banking, E-Government, E-health) are overwhelmingly governed by huge knowledge, expressed mostly in terms of business rules [BK05, OYP03]. These rules allow regulating how to do business at the intra- and cross-organizational levels alike. Restricting business activities behavior to just exchanging messages—as Web standards WSDL and BPEL adopt—represents serious obstacles to deal with such inherent rich knowledge.

**Runtime adaptivity and evolution:** To stay competitive, today's services must cater for high flexibility and adaptability. Otherwise, they become rapidly transcended by today's business market volatility and dynamism changes, where opportunistic alliances are favored. Particularly, composite services require to be dynamically adaptable to cope with different variants of requestors and their evolving requirements. Moreover, when the composition of services is flexible and dynamically adaptable, most (cross-organizational) business processes become reusable. Consequently, development efforts and costs become mastered.

**Local vs. global composition:** Any realistic service application is often composed of numerous interacting services. We argue that both BPEL-like service-focussed orchestration as well as WS-CDL-like global inter-service choreography are deemed necessary. Unfortunately, current service practices exclusively adopt one of them, with more emphasis on BPEL orchestration. We claim and demonstrate in this work, that a harmonious "local-global" service composition represents an essential milestone towards adaptive knowledge-intensive service-oriented applications.

**High Distribution and Mobility:** Most of current standards for service composition do not cope with decentralized architectures. They require that all business process activities and their instances must reside in one (logical and physical) location. This missing capability, intrinsically implies, on the one hand, the inability of intrinsically *concurrently* running different activities when it is possible. On the other hand, distribution enhances the migrating of services and their activities in accordance with the target (user's) location and computational resources. Indeed, with the increasing popularity of mobile devices like PDA and mobile phones, addressing mobility is becoming more than a commodity.

The direct technological ad-hoc deployment of such intractable advanced services may lead to serious pitfalls, limitations and unnecessary costly and risky investments, by organizations acquiring them. Indeed, even with respect to simple services, the promise of service technology in delivering, by its own adaptive composite process-centric services, is still a far-reaching objective. Adding such multi-concern requirements (e.g. knowledge-centricity, distribution, adaptability and mobility, harmonious local / global composition) is just meaning more inflexible hard-coded services, impossible to build let alone compose and adapt. The difficulties of uniformly and coherently addressing the afore-mentioned service advanced characteristics, have resulted in deployment infrastructures focussing at most one or two issues while ignoring the others.

In response to this unsatisfactory state-of-affair, we are thus witnessing a strong consensus. This technology must be embolden and steered by prior and deep understanding and conceptualization of such advanced service requirements. Only afterwards, one should addresses deployment techniques, which accurately and gradually mirroring such validated and verified domain-level service conceptualization, while reshaping available service techniques in consequence. Indeed, most of the above crucial service requirements such as knowledge-intensiveness and inherent flexibility and adaptability are *by excellence* domain-level issues. We are avoiding to vainly enforcing them through syntactical "codification" while losing their essence. Instead, in this work we endeavor to faithfully eliciting, understanding and certifying them at the business-foundation levels. As crucial advantages, we may cite, the direct involvement of all stake-holders (e.g. managers, analysts, developers, users and finally programmers) in the development. Besides that, tailored semi- and formal techniques ensure a high-level abstraction for flexibility and reliability (through validation and certification).

We aim thus at leveraging the service paradigm from its dominating technology-dependency towards more stepwise service engineering life-cycle development. In this strived life-cycle, early phases of business requirements elicitation, modelling and certification become the driving forces. Furthermore, the formal framework we are envisioning must intrinsically supports the above advanced service features (e.g. persistency, flexibility, knowledge-intensiveness, and full distribution).

After motivating the general context and research scope of this work, namely foundation and adaptability in the service paradigm, the remaining sections of the introductory chapter are organized as follows. In the second section, we summarize the main envisioned contributions of this

work, and how we aim at addressing the challenging issues while developing realistic flexible services. In particular, we shed some light of the stepwise approach and its underlying service-oriented formalism, for progressively engineering rule-intensive adaptive service-oriented applications. This chapter is then wrapped up by highlighting the content of the remaining chapters.

## 1.2 Main envisioned work's results

Along all this work, we have been indeed taking the above motivations and objectives as a roadmap for our investigations. To recapitulate on the work achieved contributions, Figure 1.1 illustrates the envisioned progressive approach. That is, for reliably engineering dynamically evolving knowledge-intensive service-oriented applications, the forwarded approach is methodologically composed of five main phases.

**UML/Business-rules for service requirements:** First, the service application structural features are semi-formally expressed in terms of stereotyped UML class-diagrams [OMG05]. Then, we describe all related intra- and inter-organizational business rules [WKL03] governing behavioral features of involved basic and composite services. We do so by respecting the Event-Condition-Action (ECA) paradigm. Furthermore, capitalizing on the strengths and discipline of architectural techniques and their transient connectors [SG96], we leverage such informal ECA-driven business to the service interconnection level.

**Service foundation and validation:** This phase is decisive as it precisely and concisely define all functionalities and behaviors of involved service components and their interactions. It should further formally validate them against misconception, misunderstandings and conflicts. We forward a service-oriented Petri nets variant called CSRV-NETS, to gradually reflect all structural and behavioral features of services from their semi-formal previous phase. It inherently addresses: Distribution, persistency (stateful), conversation and complex structuring mechanisms. For a true-concurrent operational semantics with rapid-prototyping capabilities, we are semantically governing CSRV-NETS behavior using Meseguer's rewriting logic through a tailored rewrite theory and enrichment of its MAUDE language [CDE<sup>+</sup>07].

**Synergical complementarity orchestration / choreography :** We propose to independently specify and validate any CSRV-NETS service behavior. Afterwards, we give the designer the ability to compose such validated services. This composition is achieved at the global choreographical-level, with a harmonious complementarity with CSRV-NETS (orchestration-level) services. This service composition is knowledge-intensive, driven by ECA-driven architectural business rules at both the intra- and inter-service levels. Thereby, we are enhancing service *behavioral* adaptability, both for elementary services and composite services.

**Runtime Service Adaptability:** This phase allows to endow the CSRV-NETS conceptual model with an adaptability-level. This level is based on leveraging CSRV-NETS with aspect-oriented



mechanisms [Kea97], so that runtime adaptivity of different service behaviors are achieved in a consistent and increment manner. At that adaptability-level, any business rules can be dynamically manipulated (i.e. added, removed and/or adjusted) independently of the running service components. Such adaptable ECA-driven business rules can be then dynamically woven on running behavioral service components.

**Compliant .NET aspectual environment:** Although the work does not focus that much on this implementation phase, we developed a tailored .NET environment that resumes on all the previous intuitive and founded phases. In particular, we demonstrate how to derive rule-centric aspectual .NET-based service components, where both orchestration and choreography can be performed. In this environment the governing rules are separately conceived as XML-based ECA-driven rules using RuleML-like syntax [TWB03]. Furthermore, in compliance with the founded phase, such rules are conceived as aspectual advices and dynamically (un)woven on respective running service components.

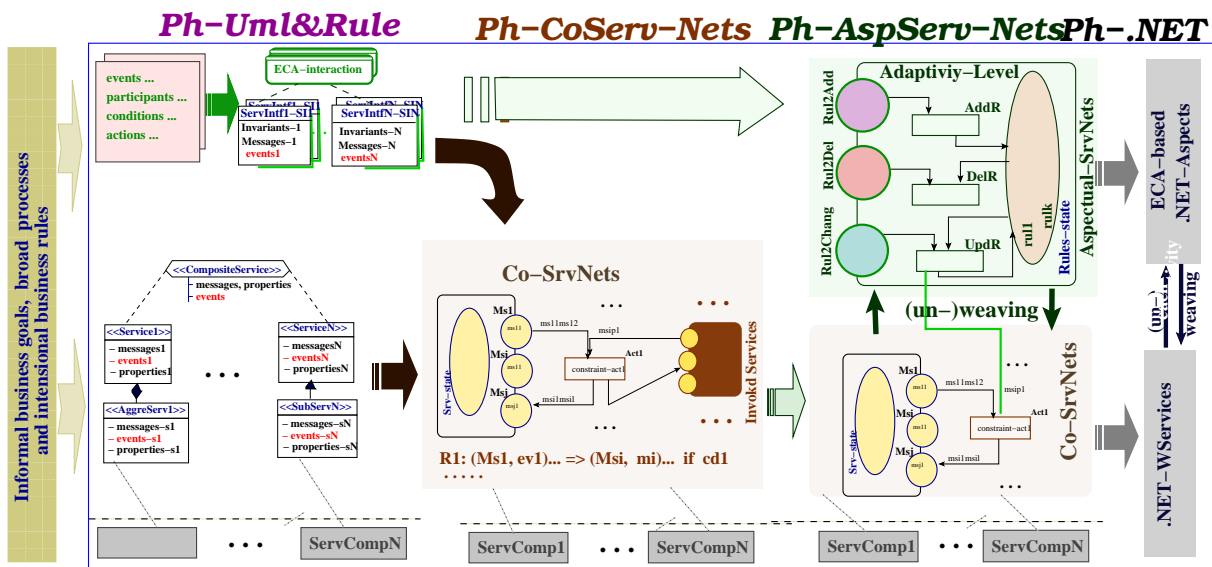


Figure 1.1: On the Disciplined and Stepwise Engineering of Adaptive and Complex Service-Oriented Systems

## 1.3 Work Outline

In accordance with the motivated objectives and envisioned contributions of this work, we overview in the following the remaining chapters by summarizing their content.

**Chapter 2: Web-Services Foundation and Adaptability: Survey and Criteria.** This chapter aims at paving the road to the main topic of this work, namely the foundation and adap-

tivity in the service paradigm. First, we provide the reader with all service backgrounds, so that (s)he can smoothly follow the subsequent main chapters. Second, we survey different proposals based on High-level Petri nets bringing formalization to Web-Services. Third, we survey recent approaches to boost the service paradigm and its Web-Services with the required adaptability, by focussing on those based on business rules and aspect-oriented techniques. We wrap this chapter, by proposing a set of exhaustive criteria allowing to compare and assess such proposals to service foundation and adaptability.

**Chapter 3: Rule-centric Stepwise Development for Service Systems.** This first main chapter motivates and presents a stepwise and disciplined approach for developing adaptive service-oriented applications. The crucial conceptual phase of this approach is based on the proposition of a tailored variant of (high-level) Petri Nets, referred to as CSRV-NETS. In this variant a special emphasis is put on the role of ECA-driven rules, service-behavior, distribution and understandability. With respect to scalability and understandability, we demonstrate through the travel agency case, how starting from a stereotyped UML-based informal description, we smoothly shift to CSRV-NETS formalization. The formalism is semantically governed by a tailored rewrite theory in rewriting-logic, and formally validated by accordingly extending the MAUDE language inherent to this logic.

**Chapter 4: Collaborative Services—Choreography meets Orchestration.** In this chapter we propose to leverage the introduced CSRV-NETS framework and its inherent methodology, so that we can cope with the global choreographical perspective, while collaborating ECA-driven adaptive services. The objective of this chapter consists thus in forwarding a sound extension of CSRV-NETS, so that a *harmonious* complementarity between the local service-focussed orchestration perspective and global inter-services choreographical perspective is achieved.

**Chapter 5: From Design- to Runtime adaptive composite services—Foundation and Deployment.** The purpose of this chapter is to go beyond the design-time adaptability of behavioral service features. We thus soundly extend the conceptual model, by endowing it with an aspectual-level. ECA-driven rules, at this CCSRV-NETS aspectual-level, can be independently and dynamically manipulated. For the dynamic (un-)weaving of such business rules on service components, we propose tailored inference mechanisms. For the formal validation, the chapter proposes a tailored aspect-oriented MAUDE-based implementation. Furthermore, we propose a .NET environment based on advanced Web-Services and aspect-oriented techniques for efficient deployment.

**Chapter 6: Conclusions and Future work.** This last chapter first recapitulates on the achieved contributions. It also discusses alternatives towards extending this work both on the conceptual and deployment tracks.

---

It is worth mentioning that an extended appendix is devoted to make this work self-contained. First, the essentials about algebraic techniques are summarized, then main concepts about (high-level) Petri nets are recalled. Rewriting techniques are then surveyed, followed by overviewing the essential about rewriting logic. Finally, since we have been using MAUDE and its reflection capabilities, the MAUDE-language main constructs are surveyed and illustrated.

## Chapter 2

# Web-Services Foundation and Adaptability: Survey and Criteria

The objectives of this preparatory chapter are fourfold. First, we overview the main ingredients of the service technology and its underlying architecture and Web-Service standards. Second, we survey most of ongoing formalizations for Web services, with a special focus on those based on (High-Level) Petri nets. Third, since this work is mainly about service adaptability, we report on different related proposals, by emphasizing the two dominating directions, namely business rule-driven and aspect-oriented approaches. Third, towards benefiting from all strengths while pinpointing serious shortcomings of this state-of-art about (Petri Nets based) rigor and adaptability in service-oriented applications, we further propose well-studied criteria to deeply assess and compare the capabilities of the surveyed approaches and proposals. The overall objective of the chapter is thus to come up with clear ideas on how should we envision an innovative approach, that is able to inherently integrate and promote all potentials of this state-of-art and overcome its serious limitations.

After a sketched overview of the service technology and its enabling Web-Services and architecture, the next section surveys most of recent work about the foundation of Web-Services through different variants of (High-level) Petri Nets. In the third section, we tackle the adaptability of composite Web-Services, by summarizing most of ongoing related approaches. We do so by categorizing the related state-of-art around its two dominating classes: Those based on the flexibility of *business rules* [WKL03] and those exploiting the strengths of *aspect-oriented* mechanisms [Kea01], for dynamically weaving concerns on running entities. In the fourth section, we put forward a set of "Web service-based" *criteria*, featuring the most important characteristics and associated requirements to meet while developing advanced composite and adaptive service-oriented applications. We then apply these criteria on the surveyed proposals and approaches. As we pointed out, this will support us in reshaping the right direction to follow, so that we result in leveraging the strengths of existing proposals while circumventing most of their shortcomings. Finally, we wrap up this

chapter, by recalling some efforts about service formalizations beyond the Petri nets direction as well as reporting on some ongoing methodologies for the stepwise development of service-oriented applications.

## 2.1 SOA and Web-Services: Overview and main Ingredients

Service-oriented computing (SOC) [Pap07] represents the best emerging technological innovations, towards faithfully automating distributed (inter-organizational) applications. In this sense, SOC treats distribution, interaction, loose-coupling and heterogeneity as main driving principles. Web-services (WS), as the main enabling of service-oriented architecture (SOA), are platform-independent self-contained software entities, with explicit *interfaces*. Web-Services are adequately tailored to be universally described, published, discovered and more importantly composed over the Web. Specifically, service *composition* allow building large-scale evolvable business processes.

Web services (WS) are characterized as *network*-addressable software units (e.g. components, modules, programs). As such, they are thus mostly developed to be used on the internet. Moreover, and in contrast to other internet-based applications (e.g. Web-Sites), WS are exclusively accessible using well-defined and explicit *interfaces*. Such interfaces are required to be easily and universally exposed, invoked and more importantly composed to reflect any complex service-oriented (business process) application, involving several basic services. Nowadays, WS is widely known by its rich package of interoperable technologies and standards (e.g. SOAP, UDDI, BPEL, and UDDI) [ACKM04]. As technology, Web-Services can thus be manipulated (e.g. described, published, discovered and composed) using adequate XML-based standards, including WSDL, UDDI, SOAP [Pap07], WS-BPEL [CGK<sup>+</sup>04] and WS-CDL [KOMC04]. These standards are rapidly gaining in maturity and acceptance. Thereby, increasingly emerging world-wide cross-organizational alliances are embracing this service technology, for automating their inherently networked business information systems.

What regards Web-Service definition, still no commonly agreed-on single definition is available. Instead, different definitions are used depending on the features to emphasis, among them we can mention the followings. In [CGS01], Web-Services have been defined as application-oriented software using specific Web standards while serving application-to-application business processes. In [New02], a web service is an interface that describes a collection of electronic operations that are network accessible through standardized XML messaging. It provides a set of functionalities to business and individual and enables universal accesses to these functionalities. On the other side, Sun-research defines WS as "services offered through the web, where typically any business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response" [Boo04a]. IBM defines WS as "an application integration technology that can be successfully used over the Internet" [Web08]. Last but not least, a commonly referenced definition by the World Wide Web Consortium [Boo04b] sees

a Web service, as a software system identified by a URI which is 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).

### 2.1.1 The Underlying Technologies for SOA

In this section, we first recall the Extensible Markup Language (XML) as it represents the backbone of SOA and WS documents. Furthermore, we go through the so-called communication-stack. Finally, different standardized languages used for describing and composing of Web services are outlined.

**Extensible Markup Language (XML).** XML [Hun04] is an extensible markup language for documents containing structured information. It provides all means to define, store, transmit and exchange (tagged) information, in a standardized and universal format ready for exchanges via the heterogeneous Web. It is important to realize that XML is a "language" on its own, 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 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 attributes. Moreover, XML allows authors of documents to define their own tags and own document structure tailored for the specific purpose.

**Web Service Communication Stack.** Web services communication stack is a collection of standardized protocols and application programming interfaces (APIs) that let individuals and applications locate and utilize Web services.

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 these layers are loosely-coupled components that interact with one another via standard protocols which have standard interfaces. The web service stack is divided up into three mains areas: communication protocols, service description and service discovery as shown in figure 2.1. The foundation layer of the Web services stack is the network layer. Web services have to be available over a network and be invoked by a service requester. The network is often based on an HTTP protocol. However this does not mean 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.

**Simple Object Access Protocol (SOAP):** SOAP is an XML-based message protocol, which is specified in a W3C specification. Moreover, SOAP provides the communication framework for transporting XML-based messages anywhere across the net. Thereby, it facilitates the communication between Web services and their clients. SOAP is the preferred XML messaging

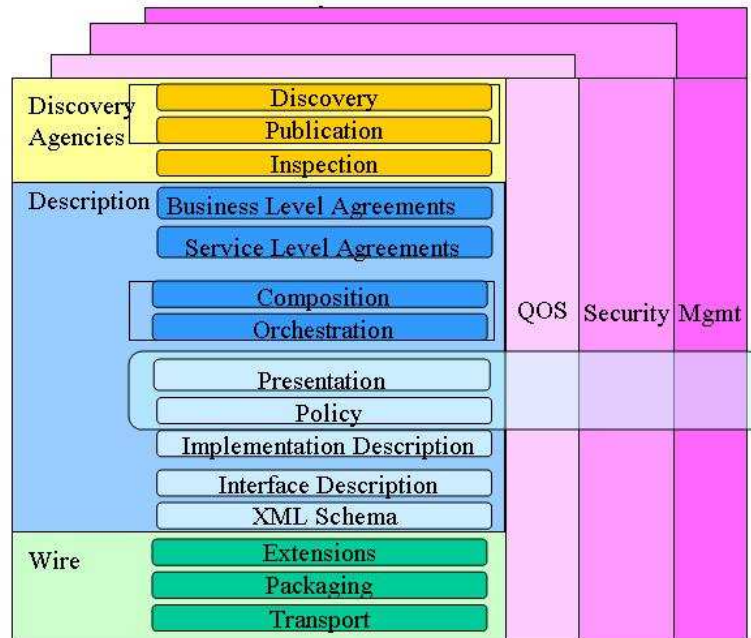


Figure 2.1: Web services stack

protocol for many reasons. First, it can be used in combination with or re-enveloped by a variety of network protocols such as HTTP, FTP and SMTP. Second, it is the standardized enveloping mechanism for communicating document-centric messages and remote procedure call (RPC). SOAP is an XML protocol that facilitates the publish, find, bind, and invoke operations. A SOAP message mainly consists of the following elements:

- **Envelope:** A required Envelope element that identifies the XML document as a SOAP messages. It contains an optional header and mandatory body.
- **Header:** An optional Header element that contains meta-information about the message, such as routing, security ,transaction management etc.
- **Body:** A required Body element that contains the actual payload of the message. ( e.g. call and response information) encoded as XML.
- **Fault:** An optional Fault element that provides information about errors that occurred while processing the message

The 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/serialization portion defines how objects are to be encoded or serialized 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/serialization features are mainly used in conjunction with the RPC (remote procedure calls[ST01]) mechanism, as explained next. RPC is used for making a request message (procedure) or function calls to a server node, and receiving the responses back. In other words, SOAP RPC defines the ability to use the SOAP protocol to execute 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.

**Web-Services Description Language (WSDL):** Web Service Description Language ( WSDL) is the layer above XML-based messaging which is a specification that describes available Web services to requesters. These descriptions written in XML form, describes the public interface and implementation of Web services. Businesses can use WSDL to advertise and then expose their services by publishing them in the registry UDDI. 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.

**Universal Description, Discovery and Integration(UDDI):** UDDI is a XML-based global, public or private online directory which enables business or individuals to list businesses that they provide as services or be discovered by other services around the global. It is responsible



for indexing Web Services, so that their WSDL descriptions can be located by development tools and applications. UDDI communicates through SOAP and acts as a directory for storing information about Web Services. The UDDI XML schema defines four types of information in order to use a partners' Web service. These types are: business entities, business services, binding templates and Models. Business entities describe information about business including their name, description, services offered and contact information. Business services provide 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, the model describes the particular protocol or standards a service uses. A registry can be executed 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.

### 2.1.2 Services-Oriented Architecture (SOA)

Traditionally, there are a number of architectural styles to build and evolve distributed systems, such as two-tier client-server, DCOM (Distributed Component Object Model), peer-to-peer and the J2EE three-tier model [YK04]. Nowadays, current trend in the application development is shifted towards loosely-coupled and browser-based applications. Therefore, HTTP is becoming one of the communication transport protocols to many of the distributed computing problems and as the future for electronic businesses. The latest evolution towards such loosely-coupled architecture is undoubtedly the Service Oriented Architecture (SOA) [ACKM04]. (SOA) is an architectural style which has been proposed recently, with promising functions to internet-based business applications. It captures software functionality as services, which can be described, located and invoked over the network.

The service-oriented paradigm for software development over the internet is thus governed by the so-called "*triangular*" service-oriented architecture (SOA). With respect to this generic architecture, Web-Services represent the most adopted and practical instantiation of SOA, but it is not obligatory the only way of using service technology. As depicted in the simplified Figure below (Figure 2.1), SOA is based on three main principles: Publish-Find-Bind. That is, following the SOA architecture, any software to used has to be published (not obligatory over the internet), where subscribers can invoke it and finally bind it to others to build complex composite services.

The Service-Oriented Architecture describes three basic roles: Service provider, service requester and service broker; and three basic activities: publish, discover, bind and invoke [Kre01].

**Service provider:** From a business perspective, this is the owner of the service. From an architectural perspective, it allows defining the service details and then publishes it to one or more repositories (service registry), based on the UDDI standard for potential users to locate.

**Service requester:** From a business perspective, this is a business that requires a certain functions

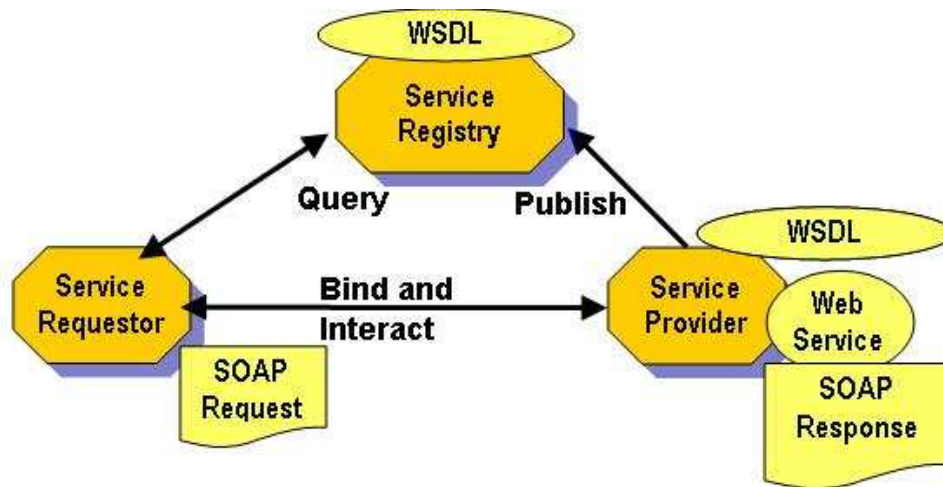


Figure 2.2: Web services architecture

that satisfy it needs. From an architectural perspective, this application permits discovering, binding and initiating any interaction within 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. It could also be another web service.

**Service broker:** This provides a searchable repository of a given 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.

### 2.1.3 Web services Specification and Composition Standards

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

Recalling first that the interaction model supported by WSDL is essentially stateless model (i.e. unaware of states in-between operation). In contrast, realistic business collaborations require long-running interactions driven by an explicit and transactional process model. Therefore, there is a need for a formal, richer description languages to fill the gap between mere interface definition languages (e.g. WSDL ) and more complex and thus realistic flow-intensive service-oriented business processes. Moreover, Web services composition languages have to support a set of minimal requirements, before the full promised potentials of Web service composition could be realized. More specifically, we require at-least five basic control patterns as described in [Aal03, SGS04] to

cope with realistic service-oriented business processes.

**Sequence:** It defines the activities being executed one after the other. That is, after an activity is finished, the next one starts.

**Parallel split:** It is a point in the workflow process, where multiple activities can be executed in parallel.

**Synchronization:** It is the case where multiple parallel activities should join together, by waiting each-other to finish, and then continuing the flow with next activity.

**Exclusive choice:** It is a point in an activity flow, where based on a decision or control data, one of several activities is to be chosen for continuation.

**Simple merge:** It is the wait for the first out of many execution paths to finish, before the flow continues.

Many specification languages have been proposed for the past years, which are aimed to support different aspects of web service interactions. These include: WSFL [Bru02], XLANG [Tha01], WSCDL [W3C04] and WS-BPEL [AB04]. This section sketches some of such Web services 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, acting both as clients and servers. It thus provides a support for any business process description. WSFL represents a business process as graphical-oriented model (flowmodel), which is a visual representation making it easy for the end-users to understand and communicate it. Graphs defined using a set of activities / tasks. A flow model is an abstract definition of the workflow process. It describes a usage patterns for a collection of a available services, so that their composition captures the expected functionalities.

A *flowmodel* 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 an involved service(s) provider(s). Next to the flow model there is a *globalModel* which defines how a given business process is implemented, namely the identity, location and the implementation of the participating services. Once the global model and the flow model for a given business process are defined, the completed business process can be exported as single WS, that is, by making them available for direct interactions to other business processes and service applications.

**Web services Business Process Execution Language (WS-BPEL).** It is the modified version of Business Process Execution Language for Web services (BPEL4WS), which has been recently defined to describe process-centric compositions of services. WS-BPEL 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.

WS-BPEL supports sequencing of peer-to-peer message exchange, both asynchronous and synchronous, within a restricted stateful interactions involving two or more parties. Using WS-BPEL, business processes can be described in two ways:

- (1) **Abstract Process:** Also called Business protocols. It uses process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, while hiding their internal behavior.
- (2) **Invocable Process:** Also called Executable business processes. It models actual behavior of a participant in a business interaction. In other words, A business process defines how a process instance coordinates the interactions with its partners.

WS-BPEL is used to model the behavior of both executable and abstract processes. WS-BPEL scope covers the three following tracks. It specify the sequencing of process activities, especially Web Service interactions. It correlates messages and process instances. Finally it allows recovery behavior, in case of failures and exceptional conditions.

WS-BPEL 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 WS-BPEL process definition provides and/or uses one or more WSDL services, and provides the description of the behavior. WS-BPEL process definition defines data variables, partners and process flow construct. A partner link type characterize 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 disadvantage and to bring more meaningful information embedded into the web content by combining semantic [TBL01] to the Web services. The realization of the so-called Semantic-Web services is underway with the development of new AI-inspired content markup languages, such as OIL, DAML+OIL and WSMO [Aa02a, Mar06]. 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 and interaction with a minimal human intervention.

## 2.2 (High-Level) Petri nets-Based Foundation for WS: Survey

There are currently several ongoing proposals dealing with this vivid area of research and practice, namely the rigorous development (e.g. specification, validation, verification) of flexible and intensively 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 objectives, formal settings and targeted application areas, it remains very hard to find a common basis for comparing them.

We will therefore restraint ourselves to only those which are very close to the aspiration of this thesis. That is, as we are envisioning a Petri Nets-based foundation for adaptive service-oriented applications, we focus this state-of-art study on those based on different variants of (high-Level) Petri nets. Moreover, for sake of readability and separation of concerns, we distinguish between service foundations based on simple Place / Transition Petri nets and those based on more elaborated high-level Petri nets.

Before presenting this state-of-art, we emphasize that towards making this thesis self-contained, we have devoted a complete Appendix (A) to recall the main definitions, concepts and properties of simple P/T Petri Nets as well as High-level algebraic Petri nets. We thus strongly advice readers, not so familiar with Petri nets and their capabilities in specifying and certifying distributed systems, to go through this Appendix before continuing this section.

Let us recalling that Petri nets [Rei91, Rei85] are among the leading specification frameworks for complex distributed systems. They enjoin several determinant characteristics. First, they introduce few concepts such as places for holding system states and transitions for capturing system functionalities such as actions and operations and their behavior. Second, they are graphical promoting more understandability (for non academic) and allowing system animation through the tokens game. Third, they are mathematically founded, with the possibilities of different semantics (e.g. interleaving, steps, concurrency) depending on the specificities of modeled applications. Fourth, 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. Last but not least, with the development of High-level Petri nets [JR91], different structured complex data can be dealt with leading to the specification/animation and verification of real-size complex applications.

(High-level) Petri nets represent therefore excellent candidate for specifying and validating and analyzing Web services. Confirmation of the claim is the growing interest and emerging approaches in that direction. Among the benefits for modelling Web-Services with (High-Level) Petri Nets, we may emphasize the followings. First, as Web-Services are by nature *distributed* applications, the concurrency behavior that characterizes (High-Level) Petri nets put them among the most suitable conceptual framework for Web services. Second, as Web services composition is driven mainly by business processes, the inherent ability of (High-Level) Petri nets in capturing different activities ordering (e.g. parallel, sequential, and-join, or-join, choice, etc.) promotes their modelling. Third,

the explicit states in terms of places in (High-Level) Petri nets enhance the specification of *reactive* and *transactional* Web service applications. These are omnipresent and represent challenging classes to specify and reason about. Indeed, reactive Web services are *stateful* and require conceptual models combining composition and conversation features. Fourth, the support of advanced structuring mechanisms in High-level Petri nets (i.e. inheritance, composition, aggregations) [BB91, Aou02, AS02], facilitate expressing complex constraints and conditions about Web services behavior.

### 2.2.1 P/T Nets-based Foundations for Web services

Depending on the goals of the formalization as well as the targeted service-oriented features and domain, we have distinguished three main approaches aiming at bringing rigor to Web-Services using Petri Nets. The first class focusses on the formalization and validation of (service-oriented) business processes. It is mainly led by the research groups headed by W.Reisig, W. der Aalst and M. Dumas, and belong to the most active and large direction. The second approach initiated by H. Hammadi focusses in particular on the composition of Web-Services, though it has also been recently extended to cope with adaptability. The third direction, we report on in this section concerns on E-Services and their B2B composition.

#### **Service-oriented BPs composition using Petri Nets [Mar03, OVvdA<sup>+</sup>07a, LMSW08].**

This approach is concerned mainly with the application of Web services to distributed, cross-organizational business processes. Each business process is conceived as an open Petri nets. An 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 modeling of the process behavior. Each process net is called 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 (called here environment) is composable with a given module net. In other words, for each input place should corresponds 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 weakly sound (i.e. each initiated process comes to a proper final state). Nevertheless, the approach does not address complex data modeling such as message arguments neither advanced business rules.

Related to this approach while using a simplified variant of open Petri nets called Workflow P/T Net (i.e. WFN) [VdABHK00], the authors in [OVvdA<sup>+</sup>07a, OVvdA<sup>+</sup>07b] proposed a systematic back and forth mapping between such WFN and BPEL specifications. The benefits include

mainly the formal analysis and *verification* of BPEL specifications using Petri Nets simulation and verification capabilities (e.g. graph-reachability, P- and T-invariants).

More fundamental questions about services and their process-centric composition (using BPEL-like standards) are recently being explored in [LMSW08]. First, a more BPEL-tailored variant of Petri Nets called *open* Workflow Nets (short oWFN) [MRS05] is forwarded. On the basis of oWFN, questions such as *compatibility* and *usability* have been re-visited. More process-centric service properties have been formally analyzed and algorithmically automated [LMSW08]. These include: (1) the notion of *controllability*, that is, the capability (or not) of a service process to interact with others, and (2) the notion of *operating guidelines*, that is, the generic characterization of strategies ensuring controllability.

Subsequently, we will abbreviate this model as WsBP-PN (i.e. *WebService Business Process with Petri Nets*).

**Web services composition and adaptation with Petri Nets [HB03].** This Place/transition Petri Nets-based model to Web services proposes an expressive net-based algebra, to capture 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. The authors first define the so-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  $(\langle \text{NameS}; \text{Desc}; \text{Loc}; \text{URL}; \text{CS}; \text{SN} \rangle)$ . With **Names** standing for the service name, **Desc** to get the description of the service functionalities, **Loc** and **URL** stand for services location and URL, **CS** is the name of the service components (in case being composite composed) and finally **SN** represent the service behavior expressed in terms of a service net.

With this Web service description as tuple, they built a rich algebra that allows combining several Web services in different ways (sequence  $[S_1 \odot S_2]$ , alternative choice  $[S_1 \oplus S_2]$ , arbitrary sequence  $[S_1 \diamond S_2]$ , iteration  $\mu S$ , parallelism with communication  $[S_1 ||_c S_2]$ , discriminator operator  $[(S_1 | S_2) \rightsquigarrow S_3]$ , refinement  $[(Ref(S_1; a; S_2))]$ , selection  $[S_1(p_1, q_1) : S_n(p_n, q_n)]$ ).

Using usual Petri Nets 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 a 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 in its 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 of current Web service technology such as BPEL and WSDL for instance. Indeed, on the one side, the proposed algebra seems to be far expressive than

these technology in terms of its richness in operators. That is, except from usual operators such as sequence and parallelism, no current Web-based language uses the so-called advanced operators like arbitrarily sequence, refinement or parallelism with communication. However, on the other side, what is clearly missing is the handling of data, such as message parameters, conditions, etc, which are among the main stones of WS languages like BPEL. Subsequently, we will abbreviate this model as PN4WSC.

Finally it is worth-mentioning that the authors of this proposal have recently tackled service adaptability. In fact, in the extension called self-adapting recovery net [HBM08], they propose how to dynamically handle exceptions and errors, by dynamically changing 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).

**E-Service Orchestration Petri Nets Model [MPP02].** This approach proposes a specific form of place/transition net for specifying business to business (B2B) composition. Instead of usual Web services terminology, the authors used an equivalent concept to *E-service*. An E-service Net is specified both in its static interfaces and in its behavior. Specifically, an E-service communicates through messages, including both the ones the E-service receives and the messages it produces. So it is more close to BPEL description but with reactive behavior. 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. Places capturing the internal behavior are modeled as usual circles and named as control places. Transitions model the behavior using these places as input/output.

Using this E-service net construction, the authors proposed then the notion of E-service orchestration net. This form of net allows composing several E-service nets, which is a specific net connecting at least two E-service Nets. It allows specifying the routing of messages and act on passing the tasks to the orchestration, from an organization to another one. For such composition they introduce a new type of places called orchestration places and drawn as hexagonal. Such orchestration places allow indicating the current organization performing the corresponding E-service compositions, 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 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 analyzing 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 behavior and their orchestration (e.g. specific place form of input/output messages, organizations, etc). However, one of the missing important issue is the relationship to current web service composition languages such as BPEL and WS-CDL. The approach also uses just black-dot tokens without advanced data structure as web-languages require. Subsequently, we will abbreviate this model as E-SvPN.

### 2.2.2 Modelling Web services with High-level Petri Nets

By exploring the state-of-art about the application of high-level Petri nets to the precise underpinning of Web-Services, we mainly found three adopted variants of high-level Petri nets, namely Jensen's Colored Petri nets (CPNets) [Jen92], Valk's nets-in-nets [Val01] and predicate Petri nets [JR91].

**CP-Nets-Based model for Web services [YK04, YK05, YTX05].** This very promising approach for Web service modeling is based on Colored Petri Nets. Recalling that CPNets [Jen92] are one of the mostly accepted and widely adopted variant of High-level Petri Nets both in academia and industry. Colored Petri nets enhance standard Petri nets with advanced primitives for the definition of the data types and the manipulations of their 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. First, it captures complex service *composition*, incorporating partners with complex conversation protocols. It permits an automatic derivation of *conversation* protocols from the composition for each involved Web service. Third, it aims at the formal validation / verification of a Web services composition and its conformance to service 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 of real-size service composition are long-term transactions, with instances 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 colored 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 color sets are declared for the messages and variables. Therefore, each color 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.
- Certain BPEL-like aspects of the composite services are still missing. For instance, compensation handling, fault handling, and message correlation are not addressed.

This composition is illustrated with a travel agency example. For instance, Airlines operations like `CheckSeat`, `ReserveSeat`, `BookSeat` or `CancelSeat` should be logically ordered and more importantly synchronized or triggered by corresponding `TravelAgency` operations such as `FindBestIterinary` and `BuildIterinary`. These `Agency-AirLines` operations have also to be synchronized with the customer operations such as `TripOrder`, `ReserveReq`, `BookReq` and `CancelReq`. The synchronization is captured by adequate transitions. The order between different operations within each service (as conversation model) are modeled using also 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 `inputPlace` (if exists) connects to the transition, and represents the reception and buffering of inbound messages for the operation. The transition also connects to an `outputPlace` (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 color sets are used to capture the protocol-relevant feature of a message.
- The synchronization 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 emphasized, 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. Firstly, although CP-Nets is a very expressive formal framework, the fact that the authors were completely bounded with the limited capabilities of BPEL, business rules governing different operations are completely missing. This is very severe drawback as business rules allow changes over time, and regulate the service composition behavior. Secondly, the fact of associating with each operation several places (and several transitions), the modeled service composition could easily become untractable 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 in a choreographical manner is completely missing. Subsequently, we will abbreviate this model as CPN-WS.

**Nets in Nets-Based model for Web services [MOO04].** This work proposes to model Web-engineering by adopting the strengths of the high-level Petri nets variant based on Nets in Nets [Val01]. 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 creating as output inscriptions and another net is destroyed from input places.

The authors exploit these advantages for modeling Web services. They put forward a four-layer 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 behavior of such service.

Important to emphasize here is that this four-based layer approach to web-engineering modeling 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, behavior, 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 domain 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 trackable manner. 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.

**Predicate-Nets-Based model for Semantic Web services [NM03].** This approach 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 languages.

The translation is very intuitive and could be highlighted as follows. Each formula is captured 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 include sequence, parallel, if-then-else, etc. Although situation calculus is very rich, it remains still very hard to be understandable by non-experts. This makes the translation to Petri nets more harder. Subsequently, we will abbreviate this model as SMWBPN.

### 2.3 Service Adaptability: Rules- and Aspect-based proposals

Although service-orientation strives for flexible composition of complex Web services, current standards simply do not promote such adaptable composition. With BPEL4WS [CGK<sup>+</sup>04] and WSCDL [KOMC04] as the widely acceptable languages for composing services, any service-oriented business process to develop must be *predefined* in design-time and remains *static* (i.e. non adaptive) during time execution.

This inability of *automatically* and dynamically adapting Web service behavior, induces severe problems for this technology to deliver all its promise. First, due to the increasing complexity of realistic service-oriented applications, any *manual* adaptation is untractable and hard, error prone and too slow to be effective. second, the lack of adaptation prevents modifying the composition behavior (e.g. activities ordering, operations, constraints, etc) at runtime, as customers or providers frequently unexpectedly change or bring fresh requirements. Third, as Web services functionalities unpredictability and rapidly change to stay competitive, static composition implies very often working on obsolete and outdate versions of static Web services. Last but not least, static composition makes very difficult dealing with optimal composition, as quality of services and optimal performance implies runtime assessment.

### 2.3.1 Business Rule-driven Proposals to Web-Service Adaptability

Business rules [KL04, RW02, WKL03] have been recognized as the main driving means towards *adaptable* information systems. Business rules can be defined as "projections of organization constraints and declarations of (internal/external) policy/conditions that must be satisfied for doing *business*". A more refined definition defines business rules [KEP00, LOS97] as: "*a set of policies for regulating the whole business within and outside an organization*". As such, they play a crucial role in determining how operational decisions within or between organizations 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. These are the dominating Event-driven conditions actions (ECA) business rules. They inform about guidelines and restrictions with respect to states and processes in an organization. Therefore their collection, expression, structuring and organization have been acknowledged as central activities within any business/software model. Business rules are further coined as the most understandable communicating means for all stakeholders.

In the last two decades, business rules have become popular in information systems (IS) [BK05] and active databases [PD99]. This is mostly because of their ability to make applications flexible and amenable to change. Due to this sensitivity for changes, business rules require *explicit treatment*. Otherwise many problems may occur, such as business logic becoming hard to maintain as being tangled within application codes or scattered over distributed partner's applications. With respect to the field of IS, several categorizations of business rules have thus been proposed. We restrict ourselves here to the two following categories, we are capitalizing on while developing rule-centric service-oriented applications.

**Intra- vrs. cross-organizational Rules:** It is highly beneficial to distinguish between those rules being internal from those externally driven. Internal rules are defined within the organization and are often derived from strategic elements that motivate their existence. External rules, on the other hand, come from the outside world. They include government regulations and laws that govern the behavior in a given industry, or rules that derive from professional practice. For service-oriented applications, external rules necessitate special treatment as emerging cross-organizational knowledge.

**Intentional vrs. Operational Rules:** Intentional rules are those seen from a general *business context* perspective. They express business laws, external regulations as well as principles for conducting business. Intentional rules are thus usually expressed in the form of *natural* language statements, referring mostly to the targeted (cross-)organization goals, the way of enforcing them (e.g. valid period, expiry date, status) and involved business processes and activities.

Operational rules are instead approached from a more pragmatic *business process* perspective. They prescribe actions on the occurrence of some business events, or describe valid states of

organization entities and resources. Operational rules usually *derive from* the translation of informal 'intentional rules' to formal rule statements. They are developed in accordance with a convenient rule *pattern*, also dealing with involved resources (e.g. *actors*, activities, activity enablers, business *entities*). For *reasoning* about operational rules, several logic-based environments exist (e.g. Jess, Prolog, Mandran) [AA02b, GLC99]. Detection of conflicts and inconsistencies belong to the main goals behind such logic-based foundations [SS99]. For practical reasons, we will thus exclusively focussing on operational ECA-driven business rules, and thus assume that prior intentional requirements rules as given.

### **Business rules and Web-Services: State-of-art**

Business rule-driven business models enjoy, therefore, very determinant advantages for coping with dynamically evolving Web-Services and related service-oriented business processes. First, they are specified independently of processes so they are intrinsically evolvable. Second, they focus on more primary business-driven requirements. Last but not least, they respect declarative descriptions, rather than specific operational ones, which opens different way of abstractly conceiving and validating them. Recalling again that all service standards for Web-Services with mainly WSDL, BPEL4WS and WS-CDL [WCL<sup>+</sup>05], are by essence rigid, static and process-centric, and thus far from being able on their own to deal with evolving rule-centric knowledge. Nevertheless, a few emerging approaches are aiming to exploit these capabilities while developing Web-Services, mainly using BPEL. We survey all of them in the following.

Papazoglou et al.[OYP03] were the first to yield the potential of business rules in service-orientation to endow BPEL with more 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 [WKL03]. 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. 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. Nevertheless, no formal verification / validation of the constructed composition is undertaken. In addition, this approach copes only with functionality-driven rules.

In [CM04], the authors present a more pragmatic hybrid approach for realizing the integration of business rules (modeled as aspects) with a BPEL orchestration. In this paper, they exploits thus their integration of aspect-oriented mechanisms [FECA04] in BPEL leading to AO4BPEL language, and capture as aspect any business rule. Such business rules as aspects are thus being woven on BPEL codes. Moreover, the approach allows business rules to be specified / evolved independently of the (standard) BPEL descriptions.

S.Dustdar et al. [RD05] put forward another promising approach combining business rules and Web-Services. In this work, business rules are conceived and externally exposed as Web-Services. Instead of WSDL, they are described using reactive RuleML [Rul05]. The rules are thus conceived as independent service *agreements* to be invoked over the Web as services. In contrast to the static WSDL, rules can thus be discovered and composed like any service, while being (internally) processed using any logic-based engine. The approach is automated with supporting tool called ViDre [NRD06]. Nonetheless, the approach does not leverage to the conceptual-level, neither copes with dynamic composition of the (WS) rules governing *activities*.

Another recently proposed approach [LLGL08], adopts Description Logic (DL) to formalize business rules governing the temporal ordering of BPEL-like activities. That is, it enforces the BPEL-workflow to stay always in compliance with the business logic, governed by business rules. Potential advantage of the DL-based semantics is the formal handling of inconsistencies and redundancies between the rules. We note, nevertheless, that the emphasis is more on the inter-activities behavior. That is, rules governing activities and their agility are not tackled. A similar but more deployment-oriented approach is proposed in [HJL<sup>+</sup>07].

Apart from these proposals that *directly* bring business rules into service technology, the confluence of Web-Services and the Semantic Web [BLHL01] could be regarded as a "radical" alternative. It permits incorporating *ontologically* interpreted knowledge in service technology. OWL-S [owl04] belongs to the leading language in that category. Since we will be focussing just on explicit business rules, semantics Web-Services remain out of the project scope.

### 2.3.2 AOP and Adaptive Service-oriented Applications

Aspect-oriented programming (AOP) was firstly forwarded by [Kea97], as the consequence to the limitations of the object paradigm in factoring out *cross-cutting* concerns (e.g. Persistence, Logging, Security, etc.). AOP allows thus extracting cross-cutting concerns from different code units (e.g. components, modules or classes) and externalizing them in so-called *advices*, as factorized encapsulated behavioral units ready to be accordingly "injected" into specific positions in concerned units. While the right positions, where these advices have to be *woven*, are referred by *joinpoints*, the different ways of combining such advices before superposing them on respective units are referred by *pointcuts*.

These main AOP mechanisms have been first implemented in the AspectJ language [Kea01]. Since then, different AOP languages have been introduced, by focussing on specific kinds of weaving such as dynamic weaving and strategies for pointcuts. What concerns the explicit handling of *business rules* as advices coupled by non-intrusive weaving, the JasCo language [DFS04] remains the most suitable. Moreover, this language has been leveraged to cope with multi-concerns in Web Services through a variant called WSML [VCJ03]. JasCo is based on two concepts: Aspect Beans for defining reusable advices as hooks and connectors for coping with different weaving strategies. At the requirements analysis, several aspect-based extensions to UML have been recently introduced

[FS07, WSHG06]. At the architectural-level [CG01], several aspectual extensions to Architectural languages (ADL) are being forwarded [GCB<sup>+</sup>06, BCG<sup>+</sup>06]. The main ideas consist in having besides usual components and connectors, new forms of cross-cutting "aspectual" components and connectors.

### **AOP and Adaptive Web-Services: State-of-art**

The first interesting proposal in this direction has been forwarded by A. Charfi et al. [CM04], as we cited above. It aims mainly at bringing more agility and modularity to the BPEL language, by enriching it with an extra aspectual level. The resulting new language named AO4BPEL [CM07], allows thus externalizing cross-cutting concerns such as security, data handling and others by separately codifying them as XML schemas and then (statically) weaving them on BPEL activities. The approach has been abstracted recently to fit any workflow language (i.e. going beyond BPEL) [CKM07].

The approach introduced by Erradi et al. [EM06] also adopts aspect-orientation, and is specifically devoted to policies and QoS concerns. The approach is not tailored to just BPEL, and it addresses *runtime* weaving of different policies on running services. An environment called MASC is supporting the approach [ETM07]. In the same line, A.Finkelstein et al. proposed in [CF05] a generic aspect-oriented language, they applied for dynamically weaving behavioral advices on BPEL code.

Another aspect-driven approach to Web-services appeared most recently in [MBM<sup>+</sup>07]. In this approach the emphasis is on the adaptability of business *protocols* while composing Web-services. Indeed, very often both syntactical and semantic mismatches occur while invoking and composing complex services. A higher aspectual level is conceived, where dynamic ordering could be resolved on-the-fly.

## **2.4 Web-Services Modelling and Adaptability: Criteria and Assessment**

In this section we endeavor leveraging the above informal and subjective surveys—about different approaches to Web-Services formalization and adaptability—towards a more *disciplined* and exhaustive assessment and comparison of their strengths and flaws. For that purpose, we are capitalizing on the conducted intensive state-of-art investigations related to different issues and concerns in the development of Web-Services and service-oriented applications in general. More precisely, first we put forwards a set of criteria reflecting crucial features of services and the requirements for their modelling. Then, we apply these criteria on the afore-discussed approaches. This comparison allow us shedding all lights on different deficiencies and strengths of any of such proposals. The ultimate results of this assessment will be playing the driving conceptual guidelines,



towards coming up in the next chapters with an innovative conceptual model, that overcomes most of the shortcomings and takes profit of the advantages.

### 2.4.1 Criteria for Web-Services Modelling and Adaptability

Towards assessing and comparing current proposals to Web-Services formalization and adaptability, we are coming up with a set of well-studied criteria to mirror crucial service features and requirements to meet to adequately handle them. For sake of clarity, we are classifying these criterions into five categories: *Service modelling*, *Service Composition*, *Practicability/expressiveness*, *adaptability and mobility*.

#### Service modelling Criteria

Under this crucial category, we refer to the capabilities of any adopted conceptual model to capture the service behavior in satisfactory manner. These capabilities should include, among others, the support of different abstraction mechanisms for taming the service complexity as well as the ability of dealing with persistency and stateful composition. More precisely, we argue the following criterions should be inherently supported by any serious conceptual service modelling candidate.

**Abstract / concrete interfacing:** As emphasized in [DD04], the explicit distinction between abstract interface description and 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 selected services, their location and costs have to be considered.

**Service interface behavior:** It is the ability to capture the behavior of the abstract service description in expressive and stateful manner.

**Concrete Service behavior:** We argue that a good conceptual model should also allow modelling the behavior 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 message parameters, operation conditions, etc is very crucial for a conceptual model to be acceptable.

**Service data structuring:** To cope 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 *service states* in the service behavior 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 enhances the service performance.

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

### 2.4.2 Service composition criteria

As widely recognized, composition is the main essence in Web services and its underlying service-oriented architecture. Composition is the ability to bring together the functionalities of more than one service (interfaces) to build a complex service and achieve thereby any realistic customer requirements and demands. Generally, Web-Services composition tackles inter-organizational services, where different organizations participate with a specific know-how and added-value.

Beyond the offered partial-ordering operators (e.g. sequence, choice, parallel, etc.) for composing business activities, we argue that for coping realistic composite services more advanced features and criteria should be available for any suitable modelling approach.

**Choreography / Orchestration:** This criterion means the explicit ability to distinguish between orchestration and choreography while composing services. Orchestration implies the ability of specifying a specific service that interacts (i.e. be composed) with other services through messages invocation, reception and replication. This restricted *single-view* composition is mainly promoted by languages like BPEL4WS and DAML-S. The choreography in contrast aims at interacting or composing several services through their interfaces in a decentralized and balanced way, without any dominating service. Technologically, choreography is particularly promoted by the WS-CDL [KOMC04, W3C04] language.

**Stateful Orchestration:** It is the ability to conceive the services orchestration instances in a persistent and reactive way. It is a crucial property in advanced and realistic services as most of them are becoming long-running and transactional.

**Stateful Choreography:** The ability to conceive the services choreography instances in a persistent and reactive way leads to the development of complex service systems.

**Stateful Conversation:** To allow composing complex web-applications, service interfaces (e.g. invoke, receive) are not sufficient. Instead a complex conversation is mostly required between different invoked and received operation services [PTDL08]. Such conversation should also be modeled in a stateful manner to cope with different instances and their persistent states.

**Dynamic composition:** Although Web services aims to generate composite services in a runtime way, existing Web services languages still achieve it only statically and manually. As pointed out in [OYP03] business rules may significantly contribute to make such composition more dynamic. As we have reported, aspect-oriented mechanisms may also contribute to adapt service composition.

**Stepwise composition:** Web services composition involves several activities including: Interface definition, interface behavior specification, orchestration, conversation, choreography and run-time adaptability [DD04]. Mastering this complexity requires a clear stepwise methodology, so that such steps can be optimally organized and coordinated.

### **Practicability / Expressiveness criteria**

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 [OMG05], and the hiding of formal technicalities and semantics as much as possible. In detail, we propose the following criterions to be fulfilled for enhancing the practicability and expressiveness of any service conceptual modelling.

**Expressiveness :** As we just mentioned an adequate conceptual model for Web services should be able to explicitly specifying service data, messages, states, etc. Moreover, to achieve a high-expressiveness most if not all criterions related to the composition and the modelling as above detailed should be met.

**Compactiveness :** Towards taming the complexity of real-size Web services at the modelling phase, the model has to be compact enough to capture large services. Refinement steps may be necessary to incrementally deals 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 WSDL, BPEL and WS-CDL are becoming the de-facto standards, any adequate conceptual model should be able to intuitively and automatically be translated into such languages.

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

**Hiding of tedious formal semantics :** The previous criterion is an important step towards hiding formal details, when clear translation steps are proposed to automatically generating 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.

	BPW-PN	PN4WS	E-SvPN	SMWBPN	2NETSWS	CPN-WS
<b>Service Composition Criteria</b>						
Choreography/orchestration	×	×	+/-	×	×	×
Stateful Orchestration	+/-	+/-	+/-	+/-	✓	✓
Stateful Choreography	×	×	×	+/-	×	×
Stateful Conversation	+/-	+/-	+/-	×	+/-	✓
Automatic composition	×	+/-	×	×	+/-	+/-
Stepwise composition	×	×	×	×	+/-	+/-
<b>Service behavior modelling</b>						
Abstract/concrete interface	×	×	×	×	×	×
Service interface behavior	+/-	+/-	+/-	✓	✓	✓
Concrete Service behavior	×	×	×	×	×	×
Data abstraction	×	×	×	✓	✓	✓
Data structuring	×	×	×	×	×	×
State intra-concurrency	×	×	×	×	×	×
Inter-concurrency	+/-	✓	✓	✓	✓	✓
<b>Practicability/Expressive.</b>						
Expressivity	+/-	×	×	+/-	✓	✓
Compactness	×	×	+/-	+/-	✓	+/-
Relation to WS technology	+/-	×	×	+/-	+/-	✓
Diagram. support	×	×	×	×	×	×
Semantics Hiding	+/-	×	+/-	+/-	+/-	+/-
Tools	+/-	×	+/-	+/-	✓	✓
<b>WS Adaptability</b>						
Rule-driven modelling	×	×	×	×	×	×
Architectural modelling	×	+/-	×	×	×	×
Runtime adaptability	×	+/-	×	×	×	×

Table 2.1: Service criteria applied on proposals for service foundation and adaptability

### Adaptability and evolution criteria

Web services are promised to achieve dynamic adaptation while composing services. We argue that the following criterions and requirements on conceptual models are essential to achieve more adaptable and runtime dynamic services.

**Rule-driven service modelling** : Business rules are the most *volatile* part of any service-oriented cross-organizational alliance [KL04]. They describe functionalities / policies and rules for doing business. The ability of explicitly modelling such business rules is therefore determinant prerequisite for making services adaptable.

**Interaction-driven Service behavior** : We argue that an explicit externalization of service composition behavior using architectural connectors [SG96], enhances by far the reasoning and adaptation such composition in a transparent manner.

**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. Reflection techniques [YM01, CCL00] and aspect-oriented mechanisms [Kea97, EFB01] represent the most advanced software-engineering mechanisms for addressing runtime adaptability.

#### 2.4.3 Service Criteria applied on the state-of-art

On the basis of the forwarded criteria for service features and modelling requirements, we assessed and analyzed the discussed approaches to service foundation and service adaptation. The following table summarizes the result of this analysis. The adopted abbreviations are as follows. We are using the symbol '✓' when the respective criterion is fully supported by the model. In contrast, the symbol '✗' is used when criterion is absent and thus not supported by the model. Finally, when a non satisfactory fulfillment of the criterion we use the symbol '+/-', that is, when the criterion is only partially supported by the model.

As depicted in this comparative table, we may easily notice that all conceptual models fail in coping with runtime adaptability in a satisfactory way. Moreover, the explicit distinction and complementarity between service orchestration and choreography is fast missing in all approaches. When it comes to the stepwise development, we also notice that little has been achieved on different available proposals. Last but not least, the disciplined handling of knowledge in terms of evolving ECA-driven business remain largely unexplored. For the other criteria, they are variably supported by different conceptual models. High-level Petri nets-based approaches, for instance, are more expressive and respond positively to more criteria than those based on simple Place / Transition

nets. Nevertheless, when it comes to the formal verification as widely-known, P / T Petri nets are more powerful.

## 2.5 Chapter Summary

As we reported in this preparatory chapter, the foundation underpinning as well as the adaptability are among the most serious challenging concerns in the emerging service paradigm. In this respect, after a general overview of main concepts of Web-Services and its underlying service-oriented architecture, we summarized different formalization based on different variants of (high-level) Petri nets. We then reported on recent research advances about adaptability and knowledge-intensiveness in Web-Services, by focusing on those based on business rules and aspect-oriented mechanisms. The chapter also proposed well-conceived criteria for characterizing crucial features of web-services as well as the requirements to fulfill them, and then assessed these criteria with respect to the discussed approaches.

Nevertheless, we should emphasize that besides Petri-Nets based proposals, other formalisms are being intensively applied to service orchestration and choreography. They include process-algebras [FGV04], temporal logics [SCZ04], graph-transformations [HHL05] and event-calculus [MS04]; just to cite few of them.

In parallel to these formalisms, methodologies supporting the development of service-oriented applications are starting to gain in maturity. For instance, the Service Component Architecture SCA [BBBea05], supported by graphical notations, aims at lifting up the service development to the business-level. A first formalization of SCA is being forwarded in [ABFL07] using architectural techniques. The so-called SOMA (Service-Oriented Modelling and Architecture) methodology [Ars04] is being promoting at IBM. Starting from business processes, the methodology derives (abstract) services and then service-components. Worth-mentioning is also the recent proposal for service requirements elicitation [JFT07]. Though it does not explicitly speak about business rules, it handles requirements with cross-organizational (event-driven) constraints. Last but not least, with respect to the widely-accepted UML method, several recent approaches have been forwarded with different goals such as the composition of services at the modelling and abstract integration of extra-functional requirements such qualities and context concerns [OH06, RIM08, MSK08, SB05]. We note here that most of these UML-based approaches to Web-Services have been recently surveyed and assessed in [MSK08].

## Chapter 3

# Rule-centric Stepwise Development for Service Systems

This chapter lies the foundational and methodological basis for developing adaptive knowledge-intensive service-oriented systems and applications. More specifically, we put forwards an advanced disciplined and stepwise approach for semi-formally analyzing and then formally specifying and validating highly adaptive knowledge-intensive service-oriented applications. As conceptual milestones, the approach capitalizes on advanced software-engineering concepts, methods and mechanisms including: (1) Intentional business rules and stereotyped UML-classes for semi-formally handling structural and behavioral service requirements; (2) ECA-driven business rules and their disciplined architectural interconnections for reflecting the evolving and adaptive interaction-centric of composite services; (3) high-level (service-oriented) Petri nets for formally specifying distributed composite services; and (4) leveraged rewriting logic and its efficient yet tailored MAUDE language for operationally. The next section brings more motivations and insights into both the envisioned formalism and its supporting methodology, and how they mostly fit the development of adaptive and rule-intensive service-oriented applications. The chapter then delves into details about the conceptual model and its earlier semi-formal ECA-UML phases.

### 3.1 Rational for the forwarded Conceptual framework

Let us first revisit the pros-arguments and potentials towards leveraging high-level Petri nets to formalize, validate and reason about adaptive rule-intensive service-oriented applications. As we reported in the previous chapter, several ongoing (High-level) Petri nets-based (HLPN) proposals are being proposed, for formalizing and reasoning about service-oriented applications and Web-Services. In the following, we first recapitulate on key advantages of HLPN as a formal setting for service-oriented applications. Afterwards, we focus on severe limitations of such existing Petri nets-based proposals. These deficiencies will indeed pave us the way towards coming up with a variant

that allows overcoming them. The envisioned CSRV-NETS service-oriented Petri Nets formalism, as well as its supporting methodology, will then be detailed in the subsequent sections.

### 3.1.1 HLPN as service foundation: Potentials and limitations

Recalling again some of the argumentations in favor of High-level Petri Nets, as a founded setting for service-oriented applications and Web-Services. These advantages include at least the following five main points:

**Understandability via visualization:** Experience shows that formalisms endowed with graphical descriptions are more accepted by *all* cross-organizational stake-holders—and not just IT-developers. Indeed, for any targeted service-oriented solution, understandability is essential in bridging the gap with the crucial business-level. Where at this level, decisive strategic goals and objectives, broad business processes and high-level policies governing any (opportunistic) cross-organizational alliance are forwarded, debated, communicated and adapted. The *aligning* of any "business-foundation" and its IT-solution relies therefore on business requirements and their understanding and handling. (High-level) Petri nets with their inherent graphical modelling and execution (token games) promote at some extent this understandability, particularly when their tedious mathematical sides are hidden.

**Concurrent and distributed behavior:** Distribution and mobility belong to the main characteristics of (advanced composite) services. Consequently, we argue that any potential candidate for service foundation requires to inherently support concurrency and distribution. High-level Petri nets are by essence concurrent, while supporting different distributed semantics.

**Type- and instance-level support:** Coping with both the type- and instance-levels represent a critical requirement to tackle service states, and so the explicit handling of persistency and conversation. We should point out that most of WS standards (e.g WSDL, BPEL and WSCDL) are not stateful, and do not thus support long-term transactional (instances of) services. High-level Petri Nets inherently support modelling at both levels, where even complex (rule-centric) service states can be specified and reasoning about.

**Validation and verification:** Service requirements validation and verification are crucial to enhance the correctness of services, and thereby attract more requestors. Validation should detect requirements misconception, missing and conflicts. As offered by High-level Petri Nets, graphical validation significantly help in this respect. Besides that, High-level Petri nets also support properties verification, using analysis techniques (e.g. P/T-invariants, siphons and traps [ABS00]).

**Service component abstraction mechanisms:** Current WS standards propose just a monolithic interfacing, where neither behavioral- nor structural-based hierarchy are possible. This



significantly hinders the tailoring of (composite) services / interfaces to specific requestors and their profiles. Object-oriented advanced mechanisms (e.g. inheritance, role and aggregation) have been well integrated in High-level Petri nets [BB91]. Since we are benefiting from our previous component-based Petri nets CO-NETS [AS02], we envision delivering complex hierarchical service components and interfaces.

As detailed in the previous chapter, most of these advantages of High-level Petri nets have been exploited, while formalizing and reasoning about Web-Services. Nevertheless, we still experience severe *limitations* hindering the disciplined development of service-oriented applications, particularly those required to be highly adaptive and knowledge-intensive. More specifically, among the serious shortcomings, we aim overcoming with our envisioned CSRV-NETS formalism, we emphasize the followings.

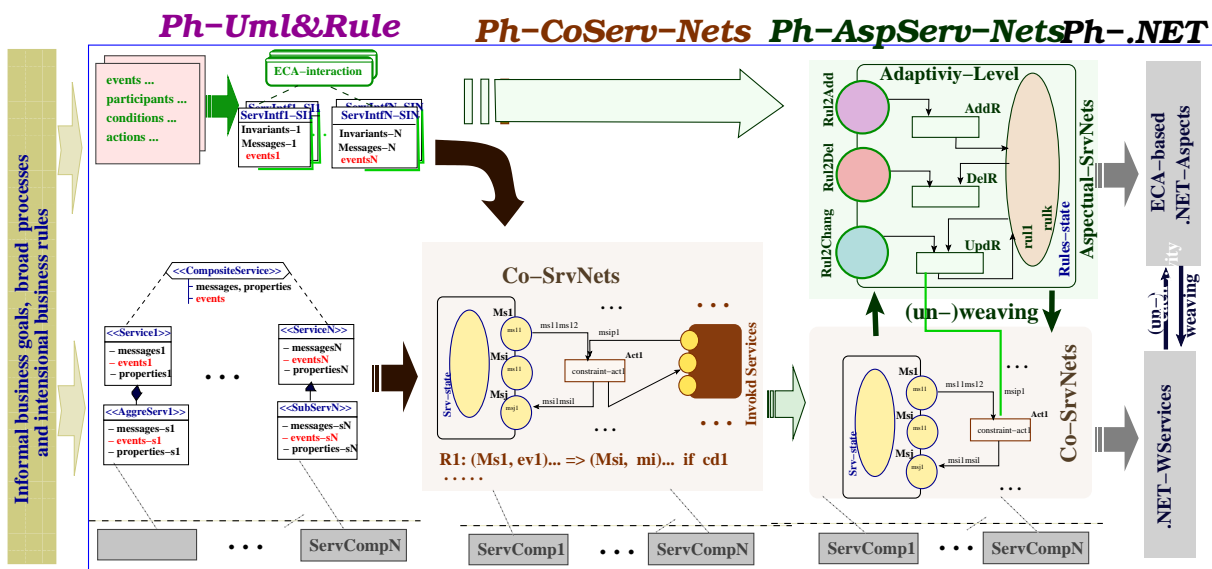


Figure 3.1: Disciplined Approach for Developing Adaptive Service-Oriented Systems

**Explicit stateful services:** Realistic services such as E-banking or E-health are mostly transactional and thus intensively stateful. Current Petri nets proposals to service foundation are instead mostly process-centric, with little to no handling of service data and states. We claim this unfortunate situation is mostly influenced by WS standards such as WSDL and BPEL. We aim thus at going beyond such standards by promoting long-running service interactions with conversational and thus stateful features. For that purpose, an advanced service states handling is proposed in CSRV-NETS. Moreover, we are inherently borrowing advanced object-oriented abstraction mechanisms (e.g. hierarchy and inheritance, roles and aggregations) while interfacing and composing such realistic services.

**Behavioral ECA-driven rules:** E-Banking, E-health and E-government as potential service-

oriented applications are inherently behavioral and rule-centric. These (composite) services are mainly governed by several evolving event-driven business rules. Unfortunately, no existing (high-level) Petri nets-based service foundation *explicitly* deals with business rules. We thus propose overcoming this severe problem by soundly and gradually integrating event-driven business rules within the CSRV-NETS formalism.

**True-concurrent operational semantics:** As we reported on, all (High-level) Petri nets-based service formalisms are governed using ad-hoc and simplistic algorithmic semantics. Besides being mostly interleaving or even sequential, such direct interpretations restrict by far the reasoning on the model. In the envisioned CSRV-NETS we go beyond such algorithmic (sequential) semantics, by proposing a true-concurrent semantics based on Meseguer’s rewriting logic [Mes92] and its intrinsic yet efficient MAUDE implementation language [CDE<sup>+</sup>07]. Besides that, with reflection capabilities of this logic, we show how to explicitly control the execution of CSRV-NETS transitions.

**(UML-centric) supporting methodology:** Existing high-level Petri nets to service foundation do not allow systematic and progressive construction of the behavioral features. This makes it very hard to novice users to build any complex service models using such formalisms. Furthermore, the tedious mathematical sides are not hidden, which leave too much confusion and rooms for different behavioral interpretations. We circumvent this serious problem by supporting the designer with explicit clear steps on how to conceive the behavioral issues, i.e. net places and transitions and their inscriptions, from UML-based structural aspects and related business rules.

**Service orchestration vrs. choreography :** Most existing founded approaches to service composition adopt the service-focussed orchestration, expressed mainly using BPEL standard. That is, the most global inter-service choreography composition is severely disadvantaged. In this work, we demonstrate that a harmonious *complementarity* between the two perspectives is very essential to promote realistic composite knowledge-intensive services.

**(Rule-centric) Service Adaptivity:** Most approaches to service foundation, including Petri net ones, do not handle adaptivity while composing services. That is, only rigid and static services may be reasoned about, compromising thereby competitiveness and opening doors to ad-hoc and risky service changes at deployment. The main objective of this work is to leverage service foundation to inherently tackle both design and runtime adaptability.

### 3.1.2 Necessity for Stepwise supporting Methodology

At the methodological-level, formalisms such as high-level Petri nets remain unfortunately still very difficult for (cross-)organizational stakeholders (e.g. managers, users, customers and even programmers). To promote the practicability and wide-usability of Petri Nets, we need thus to

hide as much as possible their tedious mathematical side. The adoption of widely-accepted semi-formal diagrammatical, intuitive yet standardized artifacts represents therefore a best compromise, while eliciting service-oriented applications.

We are therefore proposing to first describe any service structural features using stereotyped UML2.0 use-cases and class-diagrams [BJR98, OMG05]. Service behavioral aspects, on their turns, are captured through event-driven business rules [WKL03], which are inherently understandable, evolving and process-independent. Only after deriving such widely-acceptable semi-formal descriptions, we then propose to smoothly shift towards rigorous service-oriented Petri Nets specification and validation.

As illustrated in Figure 3.1, the working architecture of whole approach, we are putting forwards for developing adaptive rule-centric service applications, is composed of following progressive steps.

**UML/ECA-rules at requirements phase:** In this preliminary phase, an informal description of the targeted service-oriented application is derived. For that purpose, we assume given a priori global goals / objectives as well as broad business processes and related intentional rules governing such targeted (opportunistic) cross-organizational alliance. First, we diagrammatically express any static and structural features of such alliance using stereotyped UML2.0 Use-Cases and Class-diagrams. On the other side, we express any intra- and inter-organizational business rules following the Event-Condition-Action (ECA) paradigm [JPHL07].

**Service Nets specification / validation phase:** This phase aims at precisely, coherently and progressively defining all previous service functionalities and behaviors. The CSRv-NETS formalism is forwarded to that purpose, while promoting service distribution, persistency (stateful) and conversation. The validation is steered by a tailored rewriting logic-based semantics, we faithfully implement by extending the MAUDE language.

**Choreography and Orchestration Harmony:** CSRv-NETS behaviorally support both local service-focussed and global inter-service service compositions, also known as orchestration and choreography respectively. We should mention here that WS technology proposes two completely independent standards for these two perspectives, namely BPEL for orchestration and WS-CDL for choreography. We demonstrate how CSRv-NETS achieve a harmonious complementarity of the two perspectives on (rule-centric) behavioral issues.

**Adaptive service Nets for runtime evolution:** This phase gradually leverages CSRv-NETS with an adaptability-level based on aspect-oriented mechanisms [Kea97]. Through this evolving CSRv-NETS extension, rule-centric behavioral features are explicitly and dynamically woven on running service components and interfaces in a non-intrusive manner.

**.NET-based deployment phase:** Though in this work, we will not go in too much detail about Web-Services deployment phase, it is worth-mentioning that we have also been developing a conceptually compliant aspectual .NET environment [URAS09, ABS09d].

In this chapter we will focus on the two first phases as depicted in Figure 3.2, in this approach working architecture towards a disciplined engineering of adaptive knowledge-intensive service-oriented applications. More precisely, the remaining sections will be structured as follows. In the next section, we overview and illustrate the early semi-formal phase, where we describe both structural and behavioral features in any aimed service-oriented cross-organizational alliance. In the third section, we progressively refine the UML-based into a precise CSRV-NETS structural features. In the fourth section, we focus on CSRV-NETS behavioral features, we gradually derive from any already extracted ECA-driven rules. In the fifth section, we develop the rewriting-logic based operational semantics, where an expressive CSRV-NETS rewrite theory is proposed. The sixth section proposes a compliant extension of the MAUDE language that implements this rewrite theory, allowing thereby symbolic validation and formal verification of CSRV-NETS services.

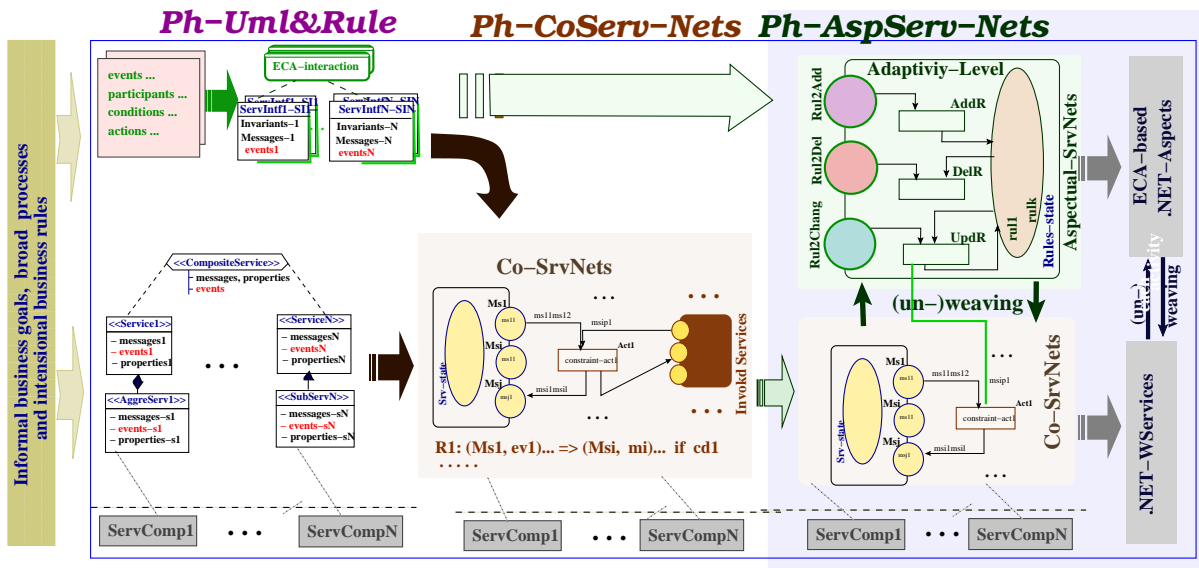


Figure 3.2: The approach two first phases, UmlRule--CSvr-Nets, as chapter focus

## 3.2 The UML-ECA-based semi-formal services description

We should first emphasize that for illustration and proof-of-concepts, we will be subsequently adopting a typical variant of the travel agency. In its simplistic case, a travel agency offers airline tickets and accommodations to its customers. Any travel agency needs thus to establish cross-organizational business links, including at-least airlines and hotels and financial institutions (e.g. bank or credit-card) for payment purpose. Further optional facilities such as sight attractions, car-renting and others may enrich a given vacation package. The customers present thus their requirements in terms of package option, period, member number, maximal-cost, location, etc. The travel agency service enters into interactions with its partners and proposes tailored offers.

Once specific offers are accepted by respective customers, the agency proceeds then to confirmation and any other required formalities. Not that customers can always cancel any offers, even when confirmed, against measured penalties. Refunds or alternatives should also be foreseen in case the agency fails performing a specific package.

This *flow*- or process-centric description is mostly followed, when (semi-)formally modelling and WS-standards deploying of Web-Services. For instance, with respect to the UML method which interests us here, activity diagrams have been the driving forces in capturing such flow-centric Web-service composition [DGS04, MSK08]. Formal techniques like Petri nets also fall in this flow-centricity in composing services, as they mostly aim at reasoning about BPEL-like processes [OVvdA<sup>+</sup>07a].

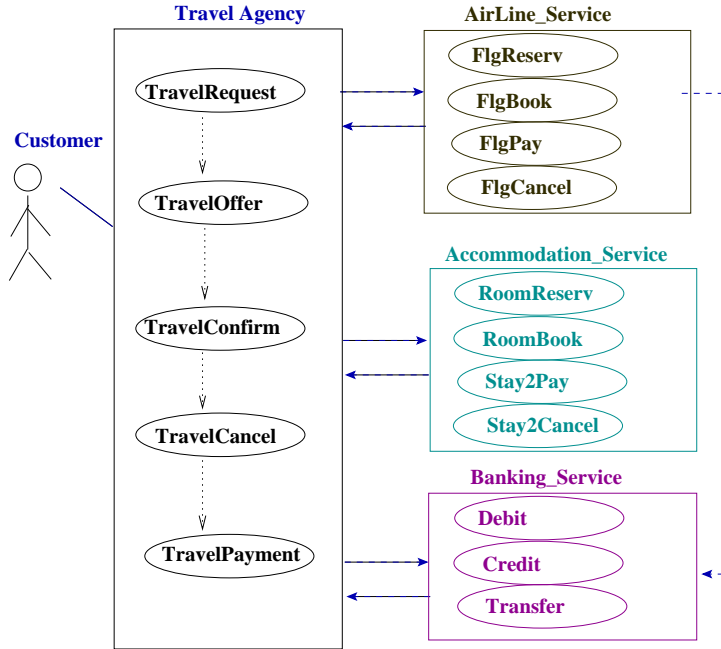
As we already emphasized, focusing only on service flow severely impedes any handling of adaptable and stateful-conversational knowledge-intensive composite services. For instance, with respect to the travel agency, by focussing only on the flow, it becomes hard to speak about evolving customer profiles (e.g. silver, golden or normal) and related added-values and competitive benefits (discounts, special offers, etc.). Moreover, with the absence of business rules governing activities (e.g. request, offer, payment, cancel, etc.), the resulting service composition is static and rigid. Besides that, the flow itself cannot be updated or tailored to the customer wishes.

Towards overcoming such severe limitations, we propose an early emphasis on different sources of knowledge and involving rules governing different business *activities*, independently of their global process. In contrast to the state-of-art, we are thus postponing the handling of the underlying process logic. This will afterwards be handled at the service-oriented Petri nets formal level. More precisely, on the one hand, we are upgrading UML class-diagrams to express different involved service interfaces, in any opportunistic cross-organizational alliance (as composite services). On the other side, we identify each required business activity and behaviorally govern it with evolving ECA-driven business rules.

### 3.2.1 Profiled UML class-diagrams: Application to the Travel Agency

A straight reformulation in terms of UML use-cases of the above informal travel agency description, could be given as depicted below the Figure on the right-side. That is, the main agency activities (i.e. use-cases) have to include: (1) The dispatching of requests to all service partners (e.g. Airline and Accommodation) once received from the customer(s); (2) The collection of returned offers and their forwarding to the customer(s); (3) The collection of any confirmation for specific offers; (4) The triggering of confirmations and collection of final bookings from the service partners; and (5) finally the handling of the payment with financial partner(s). We note that, the canceling activity could also be triggered.

Furthermore, from the Airline service interface, we require the ability of receiving flight-requests, the suggestion of respective flight offers, and the ability of booking chosen flights. The same operations should be offered by the hotel or accommodation service interface. The credit-card or any selected banking service should perform the debit, credit and transfer of the requested cost. As subsequent refinement of such general Use-Cases, we propose an *interaction-centric* upgrading of the usual UML-class diagrams.



That is, since composition belongs to the essence of the service-paradigm, we are borrowing ideas from architectural techniques [MT00] and the work in [HHL05] about UML-based graph-transformations and Web-Services. Indeed, architectural techniques allow externalizing interconnections as first-class transient connectors from service components through roles or interfaces.

As depicted in Figure 3.3, applied to the Agency case, the forwarded stereotyped UML-classes and connections can be highlighted as follows. First, all involved service components are as expected hidden units. The service interfaces are expressed as role classes. Furthermore, to facilitate service composition, we explicitly separate triggering events from received messages and invoked messages; each referred with a meaningful icon. Service interface properties (i.e. attributes) are abbreviated, and will only be detailed in next steps to help formulating any involved business rules. For instance, the customer properties can be abstracted via the name `CustInfo`. We will see later, that such `CustInfo` should at-least be composed of `name`, `birth`, `age` and `profile`.

Since the travel agency service composition is steered by the Agency service, we are stereotyping it as a connector with that special lozenge icon. In this particular case, this service connector is also to be regarded as a service-interface, thus with a hidden Agency service component. Consequently, towards composing involved services, the agency puts into force its own properties and messages. Among the required properties for a given agency, we may cite, the list of its privileged partners (e.g. airlines, hotels), list of privileged locations, targeted customers and so on.

### 3.2.2 Stepwise ECA-driven Description for Service Behaviors

As we overviewed in the previous chapter, *business rules* represent the best business and modelling ingredients in organizations to reason about knowledge and adaptation. Business rules reflect regulations and conditions for a competitive functioning of any (inter-)organization, both internally

as well as externally. As such regulations change / evolve to promote competitiveness, the governing business rules accordingly evolve [WKL03, KL04]. Business rules are mostly expressed in terms of Event-Conditions-Actions (ECA) forms, that is, on the occurrence of triggering events, related constraints should hold to perform necessary actions.

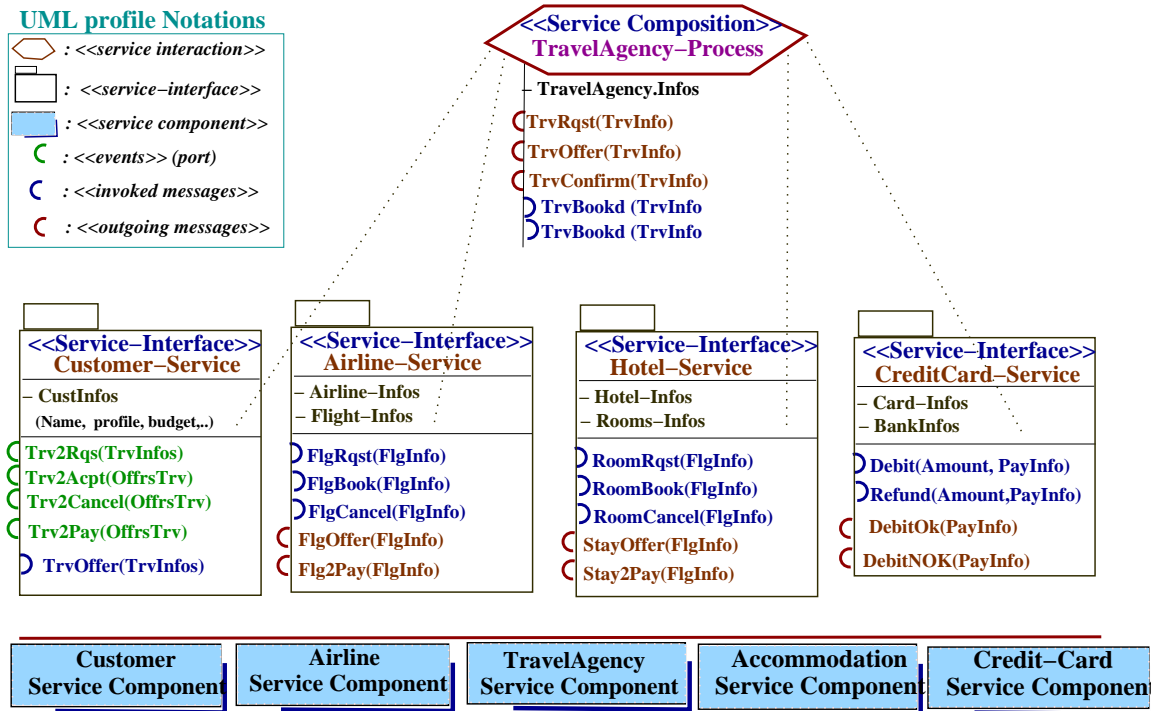


Figure 3.3: Stereotyped UML-Classes for Services Applied on Travel-Agency.

As we reported in the related-work, few proposals have been forwarded bringing business rules to Web-Services (e.g. [GS03], [CM04], [RD05]). Nevertheless, common to these approaches is that, they all focus on the deployment phase by mainly enriching BPEL and WSDL standards in ad-hoc manner. That is, existing approaches combining business rules and Web-Services do not address crucial early phases of intuitive rule-centric elicitation, formal specification and validation / verification. Furthermore, such proposals do not address the dynamic adaptability and evolution of involved rules, at both fine-grained activity and business process levels.

The envisioned approach endeavors thus to circumvent these shortcomings while benefiting from their advantages. In some details, the main features and potentials of the envisioned approach, can be summarized in the followings:

**From intuitive to disciplined ECA-driven service interactions:** Business rules belong to the main assets for supporting stake-holders in doing business. As such, they require at-priori to be discovered, described and evolved at that business-level. We aim thus at separating business rules from any (service-oriented) concrete process. Towards achieving that, we pro-

ceed in a gradual manner. For that purpose, we smoothly shift from intuitive intentional descriptive rules via more operational ECA-driven ones towards disciplined service-driven conceptualization as tailored rule-centric architectural interconnections.

- (1) **Intensional rules:** As we reported in the previous chapter, intentional rules are those seen from a general *business context* perspective. They express business laws, external regulations as well as principles for conducting business. Intentional rules are thus usually expressed in the form of *natural* language statements, referring mostly to the targeted (cross-)organization goals, the way of enforcing them (e.g. valid period, expiry date, status) and involved business processes and activities.
- (2) **ECA-driven operational rules:** Operational rules are instead approached from a more pragmatic *business process* perspective. They prescribe actions on the occurrence of some business events, or describe valid states of organization entities and resources. Operational rules usually *derive from* the translation of informal 'intentional rules' to formal rule statements. They are developed in accordance with a convenient rule *pattern*, also dealing with involved resources (e.g. *actors*, activities, activity enablers, business *entities*). We will exclusively focus on operational ECA-driven business rules, and thus assume that prior intentional requirements rules as given.
- (3) **ECA-driven architectural interactions:** Finally, a more disciplined description of these rules in terms as architectural connectors is then forwarded. The aim is to delve inside each participants and detail what specific messages, events and / or attributes and properties are required.

**Formal modelling and Reasoning:** This decisive phase is governed by the tailored service-oriented Petri nets formalism CSRV-NETS, we will detail in next sections. That is, we propose a smooth translation of any ECA-driven architectural connector, governing a given business activity or process, into a precise CSRV-NETS transition. True-concurrent rewriting rules system is then derived from these transitions. Both Formal validation through the net graphical animation and MAUDE-based rewriting computations as well as verification are then supported.

**Design- and runtime explicit adaptability and evolution:** Beyond design-time adaptability of ECA-driven rules using the CSRV-NETS formalism, we are also tackling *runtime* adaptivity by leveraging that model with an aspectual-level. As will be detailed in next chapters, ECA-driven rules are independently specified as advices at that aspectual-level. We then dynamically woven them on running CSRV-NETS service components, by either enhancing or replacing existing rules.

**Intensional rules applied to the Travel-Agency.** To stay competitive and attract more customers, any travel agency has to offer competitive and context-dependent vacation packages. Com-



petitive vacation packages should thus intrinsically depend among others on: (1) environmental situations (i.e. seasons, events, years), (2) vacation specificities (short/long, individual/group, small/ large package, etc.) as well (3) customers preferences (i.e. profile, budget, etc). In other words, to promote competitiveness and adaptability, we have to explicitly put on the foreground any appropriate business artifacts and conceptual mechanisms that dynamically handle such knowledge.

With respect to the travel-agency, at the business intuitive level, typical business rules governing its competitive functioning may include the followings. These include both the agency inter-service rules as well as intra-service rules for each of the involved partners (e.g. airline, accommodation, car-renting, attraction-service, etc.).

**BRs governing the agency composite service:** As illustration of such composite business rules, we may encounter the following:

**Rule-ag1:** When two or more persons decide for a join vacation for two weeks during June-July to specific locations, they get 20% discounts and 60% for any child. Further, when their booking is done one month in advance, they can enjoy renting-car for free.

**BRs for airline services:** : Illustration of business rules regulating flight services may include:

**Rule-air1:** When the customer books a return in the period from X to Y and (s)he is considered as golden one, his/her ticket is systematically upgraded from economy to business one. Further, if the period is more than one week and the destination belongs to specific list, a discount of 20% is applied.

**BRs for the hotel service:** Business rules regulating flight service may include:

**Rule-acm1:** For a stay more than 10 days, a discount of 10% is applied for the period from X to Y.

For the rest of this chapter, we will particularly concentrate on the Airline service. More specifically, we will illustrate all the concepts related to the first three phases on that service. That is, first, we will propose intensional rules governing the main operations of that service, namely the `flight-order`, `flight-confirmation` and `flight-cancelling`. These informal rules, will then be fitted into the ECA-paradigm and finally refined as ECA-driven architectural interactions. Subsequently, we develop on the formal contribution of these rules in the forwarded service nets CSRV-NETS framework.

A typical general intentional rule for governing the behavior of the flight service could be formulated as follows. The rule addresses all regulations about different business activities such as: `Flight-request`, `Flight-Confirmation` and `Flight-Canceling`. Obviously, to attract customers and improve competitiveness, such rule requires to evolve into different variants, depending on customers, context and other quality requirements.

*”Customers requesting for flights, beside basic mandatory information, may set their max budget to spend. Customers under 18 years get a specific discount (10 %). Confirmation of reserved flights should be done within one-week, otherwise an extra-charge is to pay within the second week. If not confirmed after two-weeks the reservation is lost with a penalty to pay (5% of the flight fare). After confirming a flight, canceling are still possible but only a percent of the flight-fare (15%) is refunded”.*

**ECA-driven operational rules: Pattern and illustration.** Towards bridging the gap between the above informal business-oriented (intensional) business rules and the service-oriented conceptual-level, we first propose to slightly reformulate and refine them into more operational event-driven rules. We thus propose to follow a specific ECA (Event-Conditions-Actions) generic pattern to express rules governing any business *activity*, within a service-oriented business process.

More specifically, we propose the following structured clauses and related primitives to express any operational business rule. First, under the clause **COMPOS-RULE** we assign an identifier to the rule while mentioning the associated governed business activity. Second, under the clause **EVENTS**, we describe all basic or composite events allowing the triggering of the rule and hence the activity. Third, under the clause **PARTNERS**, we made explicit all involved business entities (e.g. resources, actors). Fourth, under the clause **CONSTRAINTS**, we thus express all conditions to be observed and to hold to perform the associated rule. Fifth, the clause **ACTIONS** describes the effects of applying the rule, mainly in terms of messages, operations, and eventually post-triggering events and post-conditions to be observed and performed on involved partners. Finally, we also consider the case of exceptions and fault-tolerance, that is, in case of constraints violation, we prescribe which exception-actions should be performed.

**COMPOS-RULE:** *rule-identifier and respective activity-name.*

**EVENTS:** *Triggering events to enable to associated business activities.*

**PARTNERS:** *Participating business actors, entities and resources.*

**CONSTRAINTS:** *Conditions to be observed and holding to perform that rule.*

**ACTIONS:** *Actions including post-events and post-conditions to perform.*

**EXCEPTIONS:** *Concerns the actions to undertake when the constraints are violated.*

It worth mentioning to emphasize the potentials of this ECA-driven pattern for governing any business activity knowledge, towards smoothly bridging the gap with the service-oriented paradigm. First, by explicitly distinguishing any triggering events for a given business activity, we are reflecting the event-driven nature of SOA and its enabled Web-Services. Second, with the explicit description of any involved business entities and resources as partners, we are reflecting the compositional nature of SOA, where such partners will be later playing the role of service interfaces. Third, we are allowing the actions to include, as optional elements, post-triggering events and post-conditions to handle exceptions and fault-tolerant behavior. Last but not least, though we are associating any ECA-driven rule to a given business activity (i.e. business ”activity-aware rules”), and will

be detailed in next chapter, we are also tackling rules cross-cutting different activities. Since such "process-aware rules" may involve several business processes and thus several services, we have categorized them as choreography-enabling, as will thus be handled in the next chapter. In other words, the present (business) "activity-aware rules" are more suitable for service orchestration than choreography.

► **Example 3.2.1** In the previous paragraph, we presented a typical intentional business rule of the Airline service, regulating different activities such as request, confirmation and canceling within such service. Nevertheless, due to the nature of such informal language-dependent intentional rules, tedious effort should be undertaken to bring such rule to the more disciplined ECA-driven pattern. First, we should associate for each business activity at least one rule. That is to say, instead one global intentional rule, we result now into three rules, to explicitly and separately govern the request, confirm and cancel business activities. Second, for each of the resulting ECA-driven rule, we have to explicitly describe all its ingredients (e.g. events, partners, constraints and actions). More specifically, the three resulting ECA-driven rules from the refinement of the given Airline intentional rule can be detailed as follows.

The request ECA-driven rule presents no particular difficulty and is straightforwardly derived from the first sentence of the intentional rule. It remains just to mention that as involved partners, we have here not just the Airline but also the Customer who is triggering the request event.

**COMPOS-RULE:** Rrq1-RequestFlight.

**EVENTS:** The customer request for a flight.

**PARTNERS:** Customer and Airline.

**CONSTRAINTS:** The fare must be less than the customer budget.

**ACTIONS:** Do Reservation (as action and post-event) with related discount (e.g. 10% for less than 18 year-old).

Similarly, the ECA-driven rule governing the confirmation activity is directly extracted from the intentional rule as given below, with the following peculiarities. The triggering events is not just customer confirmation triggering but also the reservation event, that is, to do a confirmation one should proceed a reservation first (could be of course simultaneously). In the same spirit, the ECA-driven rule governing the canceling activity could be formulated.

**COMPOS-RULE:** Rcf1-ConfirmFlight.

**EVENTS:** The customer confirm a reserved a flight.

**PARTNERS:** Customer and Airline.

**CONSTRAINTS:** if Customer is late to confirm a penalty is assigned (after 15 days and before a month).

**ACTIONS:** Payment of the fare and of eventually of the charge.

**ECA-driven service-oriented Architectural Interactions.** From the previous informal ECA-driven description of any business activity, in any service-oriented business process, the purpose of this phase is to shift to a more disciplined yet interaction-centric conceptual modelling. Furthermore, we aim at bridging the gap with the service-oriented level, by emphasizing service interfaces and their behavioural compositions. We thus propose architectural techniques and their transient architectural connectors. Indeed, architectural techniques [SG96] permit externalizing interactions as first-class entities called connectors. They involve service interfaces and behavioural glues, reflecting different service interconnections and their composition logic.

Intuitively speaking, given an interaction ECA-driven business rule governing the behaviour of a business activity in a service-oriented business process, we propose to seemingly shifting it towards a corresponding architectural connector through the following steps:

**Service Interface properties:** Depending on the rule ingredients (e.g. events, attributes and messages) required from different service partners, we transform each involved partner or business entity name into a service interface. In the terminology of architectural techniques, it corresponds to a connector role. That is, for any partner, we precisely define its service interface as composed of all required messages, events and / or properties towards expressing the intended behaviour at the interaction level by the rule.

**Service interaction properties:** Still depending on the rule, specifically the constraint part, additional messages, attributes, constants and invariants could be defined as part of the interaction itself. In such case, the interaction could be conceived later as an independent third-party service; this could be located either at the provider side or independently conceived to govern the cooperation between the involved services.

**Service ECA-driven interactions:** We finally capture the service interaction behavior itself in a compliant ECA-driven manner. In terms of architectural terminology the rule corresponds to the connector glue. Whereas, in terms of the service-oriented paradigm, that glue captures a behavioural composition of services via their interfaces. The precise description of the rule, as we detail below, promotes the afore-described ECA-driven intuitive pattern.

More precisely, the ECA-driven architectural service interactions conceptual model we are forwarding to leverage intuitive ECA rules to a more discipline and interaction, can be summarized as depicted in Figure 3.4. First, under the keyword **ECA-Interaction**, we give a name to that service architectural interaction. Then, we define instances for participating service interfaces under the clause **participants**. For any conditions that must always hold, we have reserved the keyword **invariant**. We also consider the case where besides the interface information also proper constants / attributes are required at the interaction-level. We describe them after the keyword **constants, attributes or operations** depending on the associated case. The ECA-driven architectural interaction rule itself begins with the keyword **interaction rule**, where a name for the specified rule is given.

**ECA-interaction** <interaction-Identifier>  
**participants** <list-of-participants>  
**invariant** <unchanged interaction constraints>  
**constants/attributes/operations**  
 <extra-required elements for the interaction>  
**interaction rules:** <Rule-Name>  
   **at-trigger** <(set-of-)events>  
   **under** <conditions>  
   **reacting** <set-of-actions>  
**end Interaction**

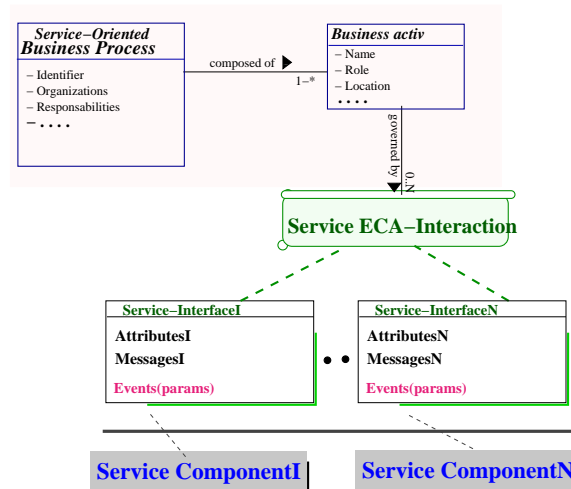


Figure 3.4: The generic ECA-driven architectural service interactions pattern

Then, we specify the triggering events of the rule under the clause **at-trigger**. The constraints to be observed are specified after the clause **under**. Finally, the actions to be performed when the triggering events and the constraints are holding are to be specified under the keyword **acting**.

As depicted in the right-hand side of Figure 3.4, this textual ECA-driven architectural service modeling will be enhanced and completed with an intrinsic graphical representation, with the following characteristics. We first attach the name of that behavioural interaction to a contract-like graphical icon. Then, for each required interface, we conceive a graphical box within it we describe its name accompanied with all required events, messages and attributes. Finally, a discounted line is then relating the contract-icon to each of the interfaces. When necessary, we also mention the involved (hidden) service components, from which the interfaces are extracted.

► **Example 3.2.2** With the support of this detailed service-oriented ECA-driven architectural interaction model, we refine in the following the ECA-driven business rules we already discussed for different business activities related to the flight service. More precisely, we consider here the architectural interaction rules related to the flight request and confirmation.

As depicted in Figure 3.5, to precisely express the request ECA-driven behavior at the architectural-level, we have first to detail all information required from both the customer and flight services, using their respective interfaces. That is, from the customer, we require as attributes the customer identifier, name and the age (to check for discount). Furthermore, it is the customer who should trigger the request for flight, through the event `RequestFlg(...)` parameterized by the wished flight and the tolerated budget. At the flight service side, we need the flight-reference and -information (i.e. departure, destination, date and time). We further need the seat-availability and reservation-list (so that we can remove / insert new ones). Finally, to keep track of successful reservations, we use the action `RequestFlg(...)`.

The interaction behavior itself starts by assigning two instances for the customer and flight ser-

```

ECA-Interaction FlightRequest-SC
participants Cst: CustomerRqFlg-SI;
               Flg: FlightRqFlg-SI;
attributes FlgRf, RsvRf: String;
interaction rule : FlightRqst-SC
at-trigger Cst.RqstFlg(FlgInfs,Bgd)
under (FlgInfs = Flg.FlgsInfs(FlgRf)) and
  ( (Flg.Avail(FlgRf) > 0) and
    ( Flg.FlgsFar(FlgRf) ≤ Cst.Bdg)
acting Flg.Add([CstId.RsvFlg(FlgRf)], RsvFlg)
Let Price = FlgsFar(FlgRf) and
if (Cst.Age < 18) then Price = 0.2*Price
Flg.Reqstd(RsvRf, Price, FlgsInfs)
end FlightRequest-SC

```

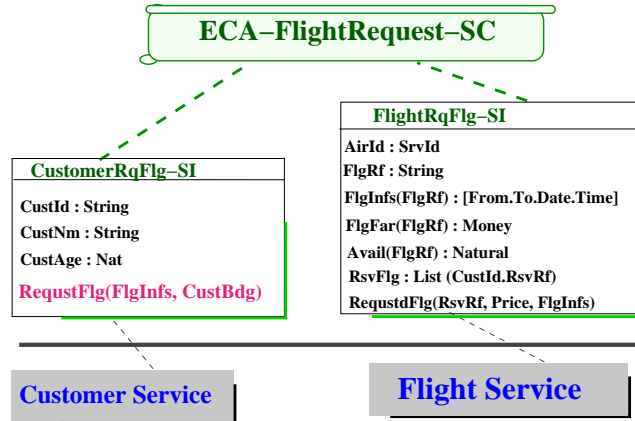


Figure 3.5: ECA-driven rule for the Flight-Request Activity

vices, namely *Cst* and *Flg*. Then as the informal rule stipulates, at the triggering of a request for flight from the customer, different constraints have to be observed. These constraints are a conjunction of the following conditions: (1) The wished customer flight information should coincide with the discovered flight ( $FlgsInfs = Flg.FlgsInfs(FlgRf)$ ); (2) There are still free seats of that flight ( $Flg.Avail(FlgRf) > 0$ ); and finally the flight fare is less than the tolerated customer budget ( $Flg.FlgsFar(FlgRf) \leq Cst.Bdg$ ). The actions to perform in this case consist of: (1) The addition of this customer to the reserved list to that flight; and the invocation of the message confirmation of that flight-request ( $Reqstd(RsvRf, Price, FlgsInfs)$ ), while considering the case of discount for young customers (i.e. 20% when the age is less than 20).

Similarly, the ECA-driven rule governing the confirmation business activity as depicted in Figure 3.6 can be precisely modelled using the following ECA-driven service architectural interaction. First, like request the confirmation involves the customer and flight services. We note that a bank or credit-card could come into play for the payment; instead we abstract it away just as a message to be send by the flight service. The triggering events are now a conjunction of the request from the customer and the presence of a triggering message that the flight has been already reserved. We then check that this flight has been really reserved (i.e. it belongs to the reservation list). We also check, whether the confirmation date is more than two-weeks, from the reservation date, but still within a month period (i.e.  $15 < CrDate - Date > 30$ , with *CrDate* standing for the current date). In such case, we apply a penalty of 10 % on the price to pay. Finally, as actions we first add this confirmation to the associated confirmation list and send a payment message (to the associated payment service such as credit-card).

```

ECA-Interaction FlightConfirm-SC
participants Cst: CustomerCfmFlg-SI;
    Flg: FlightCfmFlg-SI;
attributes CrDate: Date;
interaction rule : FlightConfirm-SC
at-trigger Cst.ConfirmFlg(RsvFlg, Date) and
    Flg.ReqstFlg(RsvFlg, Price, Date)
under ([CustId.RsvRf] ∈ RsvFlg) and
    Let Price = if ( 15 < CrDate-Date > 30)
    then Price+= 0.1*Price and
acting
    Flg.Add([CustId.RsvFlg(FlgRf)], CmfFlg)
    Flg.Pay(RsvRf, Price, FlgInfs)
end FlightConfirm-SC

```

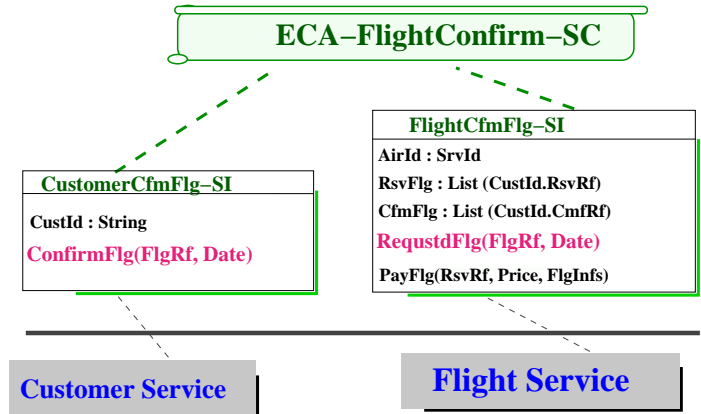


Figure 3.6: ECA-driven rule for the Flight-Request Activity

### 3.3 CSRV-NETS: Structural Features Modelling

As first step towards service foundation, we propose to precisely define different properties, events and messages to be involved in basic or composite services. For that purpose, we recapitulate on the previous phases of the stereotyped UML-classes. More specifically, to bridge the gap with the discussed service-oriented UML-based descriptions, we will first introduce the concept of service-state. Generally speaking, a service-state allows gathering different properties (i.e. attributes) of a given service interface. The precise description of such service *states* enables us afterwards, to specify the *stateful* and reactive concurrent behavior of respective services.

More precisely, we specify service-states as *algebraic* terms in the form of specific *tuples*. These service states-as-tuples allow detailing in a precise manner any generic attributes, declared in the previous UML phase. Furthermore, by adopting a MAUDE-like algebraic setting and rewriting techniques [CDE<sup>+</sup>07, Mes92], we can also reason on such state properties. Let us first informally highlight such service state-as-tuple:

- Any service state is conceived as an algebraic term of the form

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

- **SvId** is interpreted as an observed service state identity taking its values from a given appropriate abstract data type ADT (that we assume denoted as STId);
- **sv\_pr<sub>1</sub>, ..., sv\_pr<sub>k</sub>** are the observed identifiers for service state properties, which we assume having at a given time as current values respectively **vl<sub>1</sub>, ..., vl<sub>k</sub>**. 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 *Service Properties*

*Identifiers and Service Properties Values*).

- To enhance privacy, we allow *hiding* values of specific service state properties when required. To declare such Hidden service state properties, we adopt the (co-algebraic) notation of "attribute-as-functions"; so if for instance the value of an attribute identifier, denoted by  $sv_{h_1}$ , is to be hidden, we denote it as a function  $sv_{h_1}(SvId)$ , with  $SvId$  the corresponding service state identity<sup>1</sup>.
- Messages involved in a given service interface are specified as algebraic operations, including at least one service state-identifier. Moreover, messages are declared as local when acting on states only within an interface. Observed messages instead participate in a composite service interactions (being it orchestration or choreography). Finally messages may be imported from other interfaces to constraint the activity-flow as it should be in the composition. In other words, to bring more expressiveness, we explicitly distinguish between three types of messages. This allows us afterwards to adequately address their corresponding behavioural semantics.

**Local messages** : These are messages that are declared and exclusively exchanged within a given service interface. They either trigger state changes in such service interface and/or allow controlling activity-flow.

**Imported Messages** : These messages are declared in other service interfaces and used by the given service interface to facilitate defining the service process.

**Exported Messages** : These messages are declared within a given service interface and used by other service interfaces to compose services.

On the basis of this intuitive notion of service-state, we formally define the notion of (CSRV-NETS-)service state as follows.

**Definition 3.3.1 (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:  $\{Bool, STId, SPId, SP\_Value\} \subset Sv_D$ . 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 sort (so we can speak about statefull service interface).
- $\{Op\}_{ST_{Sv}}$  is a set of service state operations indexed by  $STId \times (SPId \times SP\_Value)^+ \times ST_{Sv}$ . More precisely, with each service state sort from  $ST_{Sv}$  a service state operation is associated reflecting the corresponding tuple of such service state sort.

---

<sup>1</sup>Formally, hidden attributes are regarded as co-algebraic functions[HHJT98]. That is  $sv_{h_1}$  is a function from service states to a corresponding type of the value,  $sv_{h_1} : \text{Service-State} \rightarrow \text{SP\_Value}$ .



► **Remark 3.3.2** As we emphasized, for 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:

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

Where  $SvId \in STId$  and  $\{sv\_pr_1, \dots, sv\_pr_p, sv_{h_1}, \dots, sv_{h_q}\} \subset SPId$  and  $\{vl_1, \dots, vl_p, sv_{h_1}(SvId), \dots, sv_{h_q}(SvId)\} \subset SP\_Value$

The following presents a precise description of this service state as tuple in terms of notations inspired from the functional-level of the MAUDE language [CDE<sup>+</sup>07].

```

omod 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 .
omod.

```

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 (i.e attributes) may be observed or hidden (as a co-algebraic function). We can gather all observed (resp. hidden) properties together in new sort we called  $Obsv\_part$  (resp.  $Hidn\_part$ ).

By associating messages and events to such CSRV-NETS service-state, we then results in the definition of CSRV-NETS service template specification as follows.

**Definition 3.3.3 (SERVICE-structure)** The structure specification of a given service is defined as a pair  $(Sv_D \cup ST_{Sv} \cup Ms_{Sv}, \{Op\}_{ST_{Sv}} \cup \{Op\}_{Ms_{Sv}})$  with:

- $(Sv_D \cup ST_{Sv}, \{Op\}_{ST_{Sv}})$  is a service state structure as defined above.
- $Ms_{Sv}$  is a set of ‘message generator’ sorts different from  $Sv_D \cup ST_{Sv}$ . We assume that  $Ms_{Sv}$  is composed of three sets of message sorts:  $\{Mes_{l_1}, \dots, Mes_{l_i}\}$  for local message sorts,  $\{Mes_{i_1}, \dots, Mes_{i_i}\}$  for imported ones and  $\{Mes_{e_1}, \dots, Mes_{e_e}\}$  for exported ones.
- The message operations,  $\{Op\}_{Ms_{Sv}}$  is a set of message operations, that is, operations indexed by  $STId^+ \times Sv_D^* \times Ms_{Sv}$ . Thus, with each message sort  $Mes_{ij}$  from  $Ms_{Sv}$  a message operation (denoted  $ms_{ij}$ ) is associated. Each imported / exported service message has obviously to contain at least two of its arguments of sort  $STId$  (i.e. it concerns at least two service states).

► **Remark 3.3.4** CSRV-NETS template specification as similarly syntactically described using the MAUDE notation. In this description, with  $\text{Spr}_i$  we refer to any specific *sort* for the  $i$ -th-state property. Similarly,  $\text{Sarg}_i$  refers to the sort associated with the  $i$ -th-argument of a given message.

```

omod Service-Template is
  extending Service-state
  subsort Spr1 ... Sprn Sprh1 ... Sprhm < SP_Value .
  subsort Sarg11,1 .. Sargl1,l1 .. Sargi1,1 .. Sargi1,i1 < SVD
  subsort Mesl1, Mesl2, ..., Mesll < Local_Messages .
  subsort Mese1, Mese2, ..., Mesee < Exported_Messages .
  subsort Mesi1, Mesi2, ..., Mesii < Imported_Messages .
  (* observed properties *)
    op ⟨- | svpr1 : -, ..., svpr1 : -⟩ : STId Spr1 ... Sprk → Obsv_part .
  (* hidden properties or as functions *)
    op ⟨- | svh1(STId), ..., svhl(STId) : -⟩ : STId Sprh1 ... Sprhl → Hidn_part .
  (* local messages *)
    op msl1 : STId ... Sargl1,1 ... Sargl1,l1 → Meslp .
    ... ..
  (* import messages *)
    op msi1 : STId ... STId ... Sargi1,1 ... Sargi1,i1 → Mesip .
    ... ..
  (* export messages *)
    op mse1 : STId ... STId ... Sarge1,1 ... Sarge1,e1 → Mese1 .
    ... ..
omod.

```

### 3.3.1 Application to the Travel Agency

Following this CSRV-NETS-service template, we will restrict ourselves to the formalization of the Airline service-state. The other services such as the accommodation and the Credit-Card can be similarly formally defined. We note that the agency composite service will be structurally and behaviorally specified in the next chapter. To concentrate our focus on the Airline service and its interaction with the customer, we have extracted this interaction from the complete agency UML-based description of Figure 3.3. This projection of the stereotyped UML-classes for the Customer-Airline interactions, is depicted in Figure 3.7. We note of course that this externalized Customer-Airline composition is to be hosted at the Airline service.

The corresponding Airline or CSRV-NETS flight service-state can thus be derived from this UML-based service interactions, while following the above CSRV-NETS generic template. First, we have to algebraically define in detail all required abstract data types (ADTs), involved in flight properties or message and event parameters. Such ADTs should include, among others: Dates (of departure / arrival), Cities (of departure / destination) and country names (shortly *Dest* and *Dep*), reservation and booking codes (shortly *RsvRef* and *CfrmRef*), costs for flight fare. Information about the customers such as name, address, age, members (e.g. child, infants) have also to be precisely algebraically defined (shortly *CUST\_INFO*). Data related to requested and reserved

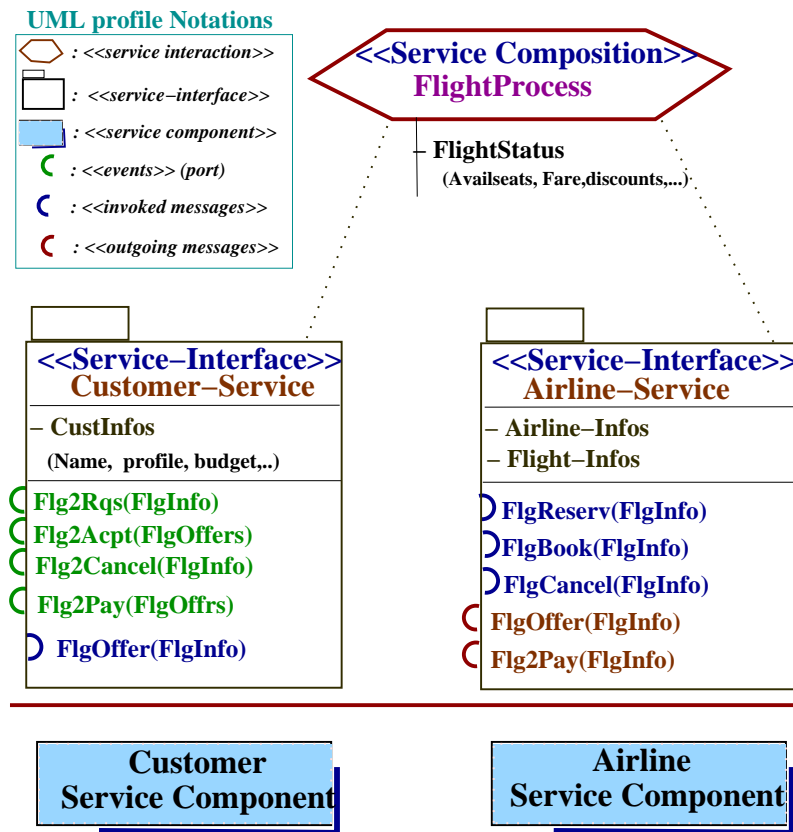


Figure 3.7: SteroTYPed UML-Classes for Services Applied on The Airline Service.

flights are also gathered as tuples (shortly RQFLG\_INFO and RSFLG\_INFO). Reserved and confirmed passengers are tracked with Ids and Refs (shortly PSSG\_RSV and PSSG\_CMFR). Worth-mentioning is that defining such ADTs is required not just for sake of precise service-states, but also for their crucial role in rigorously expressing business rules later. We gather all these required Data and refer to it as Airline-Data, which we may be precisely algebraically defined as follows.

```

omod 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

```

(\* variables to use in the service net behavior later \*)

```

vars Fg : FlightId ; 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.

```

With the support of this flight data-level and the UML-based customer-airline interactions (from Figure 3.7), we are able to precisely capture the corresponding CSRV-NETS template. We refer to the flight service-state with `Flight_St`. That flight state is identified by `FlightId` sort, is then composed of the Airline name, flight data (i.e. `depCity`, `DestCity`, `DepDate`, `DepTime`, `ArrDate`, `ArrTime`), available seats (`AvSeat(FlightId)`<sup>2</sup>), and reserved / booked passengers. Secondly, for each message declared in the corresponding UML class-diagram, we associate a corresponding message sort. For instance, to the message `FlightRequest` we declare the sort `FLGHT_RQ` and the (imported) message `FlgRq` with all required parameters (e.g. customer-info, agencyID, flight itinerary and preferences). The same reasoning is to be applied to all other messages.

```

omod 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.
  subsort FLGHT_RQSTD FLGHT_BKD FLGHT_CLD PAY_FLGHT PAY_PNLNLY < exported_Msg.
  (* AirLine State Properties *)
  op < _ | AirLId : _, FlInf : _, AvSt(FlighId), RsvP : _, CmfrP : _, DirRs : _ > :
    FlightId 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 .

```

---

<sup>2</sup>As a hidden property to be just checked.

```

op FlgClId : CustId AGCYId AirLIId BkRef → FLGHT_CLD .
op Payflg : CustId AGCYId BkRef money → PAY_FLGHT .
op PayPnlIt : CustId AGCYId BkRef money → PAY_PNLTY .

```

(\* variables to use in the service net specification \*)

```

vars Dy : Date ; Gc:AGCYId .
endo.

```

### 3.4 CSRV-NETS: Behavioral Modelling of Services

In the previous section we defined the precise CSRV-NETS service template states, through a smooth mapping from the semi-formal diagrammatical stereotyped UML-classes. This section aims at *behaviorally* leveraging such service template states, by mainly capitalizing on the governing ECA-driven business rules and their architectural modelling. More precisely, first we systematically construct corresponding CSRV-NETS places from the specified service states. Then, we detail how to derive CSRV-NETS transitions, from related ECA-driven architectural rules. Broadly speaking, the CSRV-NETS net, we propose to associate with a given service template is constructed as follows.

- The places of the net are precisely defined by associating with each service message generator one ‘message’ place. Graphically, we use colors to distinguish between events and invoked messages as well as exported messages (resp. orange, green and blue) . Furthermore, events and invoked messages will appear on the left, whereas exported messages will be gathered on the right-hand side of the net model.
- With each service state sort we also conceive a ‘state’ place. We draw them as ellipse, to highlight as much token forms as possible.
- Transitions reflect the effect of events and messages on service states. Their precise behavior, i.e. input / output arc-inscriptions and condition will be derived from governing ECA-driven architectural rules. When several conditions are to be associated within a given transition, we split it into respective boxes.

Before we delve into the formal definition of such CSRV-NETS structure, by reflecting this intuitive construction, some preliminary ingredients such as multi- and marking-terms are necessary.

#### Definition 3.4.1 Terms, Multi- and Marking-terms for services

- For any service state property sort  $Spr_i$ , we define and denote by  $T_{Spr_i}(X_{Spr_i})$ , the associated (algebraic) terms. Where,  $X_{Spr_i}$  is a set of variables for that sort (i.e.  $Spr_i$ ). The concept of service-state terms is then defined and denoted by  $T_{ST_{S_v}}(X_{St})$ , by composing as union different state properties. The set of variables  $X_{St}$  is associated with the service state sorts  $St$ . When no ambiguity presents, we uniformly use  $X$  as the union of all variables.

- Similarly, we define and denote by  $T_{Ms_{Sv}}(X_{Sarg})$ , the algebraic terms associated with service message sorts, where  $X_{Sarg}$  are variables of message arguments sorts.  $T_{ST_{Sv}}(\emptyset)$  (resp.  $T_{Ms_{Sv}}(\emptyset)$ ) with denote the corresponding ground terms (i.e. without variables).
- To subsequently capture arc-inscriptions and service states and messages as tokens, we define the notion of multi-terms over service state- and message-terms. First, we define and denote by  $MT_{ST_{Sv}}(X)$  (resp.  $MT_{Ms_{Sv}}(X)$ ) the multi-terms over  $T_{ST_{Sv}}(X)$  (resp. over  $T_{Ms_{Sv}}(X)$ ). We propose the operator  $\oplus$  to built such multi-terms. Subsequently, for sake of abbreviation, we refer such multi-terms as  $[T_{ST_{Sv}}(X)]_{\oplus}$  and  $[T_{Ms_{Sv}}(X)]_{\oplus}$  respectively.
- Finally, in order to capture any running service states, as a composition of different tokens as multi-terms, we define and denote by  $BT_{Sv}(X)$  (resp.  $BT_{Ms}(X)$ ), the multi-term over the above multi-terms. More precisely,  $BT_{Sv}(X)$  is defined over  $SvPl \times [T_{ST_{Sv}}(X)]_{\oplus}$  and  $[T_{Ms_{Sv}}(X)]_{\oplus}$ . The sort set  $SvP$  will correspond to the places in CSRV-NETS. The union operator governing such multi-terms will be referred by  $\otimes$ . (with  $\emptyset_B$  as identity). Again, we will abbreviate such multi-terms with the notation:  $[SvP \times ([T_{ST_{Sv}}(X)]_{\oplus} \cup [T_{Ms_{Sv}}(X)]_{\oplus})]_{\otimes}$ .

**Definition 3.4.2 (CSRV-NETS specification)** Given a CSRV-NETS-template specification as previously defined, a CSRV-NET specification is a structure  $(SvP, SvT, I.SvT, O.SvT, s, SvTC)$  where:

- $SvP$  is a set of (service) places such that  $|SvP| = |ST_{Sv}| + |Ms_{Sv}|$ . That is, the service place number corresponds exactly to the cardinality of sorts in  $St_{Sv}$  plus those in  $Ms_{Sv}$ .
- $s : SvP \rightarrow ST_{Sv} \cup Ms_{Sv}$  is a bijection associating with each place identifier in  $SvP$  a corresponding sort from  $ST_{Sv} \cup Ms_{Sv}$  as we informally commented.
- $SvT$  is a set of (service) transitions different from the place identifiers ( $SvP \cap SvT = \emptyset$ ).
- $I.SvT : SvT \rightarrow [SvP \times ([T_{ST_{Sv}}(X)]_{\oplus} \cup [T_{Ms_{Sv}}(X)]_{\oplus})]_{\otimes}$ . That is,  $I.SvT(t)$  captures all *input* arc-inscriptions (multi-terms) to a given transition. That is, by taking  $t$  as the transition,  $pi$  as input places to  $t$  with  $mt_i$  as respective arc-inscriptions, then the function  $I.SvT$  can written as  $\otimes_i(p_i, mt_i)$ . Here we assume the *sort coherence* condition, that is,  $mt_i \in [T_{s(p_i)}]_{\oplus}$ .
- $O.SvT : SvT \rightarrow [SvP \times ([T_{ST_{Sv}}(X)]_{\oplus} \cup [T_{Ms_{Sv}}(X)]_{\oplus})]_{\otimes}$ . That is,  $O.SvT$  captures *output* tokens with their corresponding places.
- $SvTC : SvT \rightarrow (T_{ST_{Sv}}(X) \cup T_{Ms}(X))_{bool}$  is a function associating a boolean expression over  $(T_{ST_{Sv}}(x(t)) \cup T_{Ms}(x(t)))$  with every transition  $t \in T$ ; where  $x(t)$  is the set of variables occurring in  $I.SvT(t)$  (which as usual should include those in  $O.SvT(t)$ ).

The concept of a CSRV-NETS service component, as society of service states and messages, can be deduced from this definition. We first need the notion of a *marked* CSRV-NET.

**Definition 3.4.3 (marked CSRV-NET)** A marked CSRV-NET is an CSRV-NET with a function  $SvM : SvP \rightarrow [T_{St_{Sv}}(\emptyset) \cup T_{Ms}(\emptyset)]_{\oplus}$ . That is, with each CSRV-NET place, we associate the current

service state or message instances.

From this notion of marked CSRV-NETS, it is now possible to define the notion of CSRV-NETS *state*. It allows capturing the distribution of markings over different (state and message) places.

**Definition 3.4.4 (CSRV-NETS-state)** Given a marked CSRV-NET as defined above, a CSRV-NET state is an element of  $[SvP \times ([T_{St_{Sv}}(\emptyset)]_{\oplus} \cup [T_{Ms}(\emptyset)]_{\oplus})]_{\otimes}$ . More precisely, by denoting such state by  $\mathcal{M}_{st}$ , it can be written as follows:  $\mathcal{M}_{st} = \bigotimes_{p_i \in SvP} (p_i, SvM(p_i))$ .

**The Airline service behavior as CSRV-NETS.** With respect to the above CSRV-NETS flight service template., the application of these behavioral constructions results in the following flight CSRV-NET behavioral service model as depicted in Figure 3.8. As defined above, for each service state and message sort a corresponding (typed) place are conceived. That is, to the service state sort **Flight** 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 side, 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-Booking**, **Flight-Cancel** correspond three associated sort *messages* places. Also, for the five invoked (exported) messages places, namely **Flight-Requestd**, **Flight-Booked**, **Flight-Cancelled**, **PayFlight** and **PayPenalty** 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 **FlightOP-Err**. As we will explained subsequently this place receive all attempts for violating the business rules related to different message functionalities.

### 3.4.1 CSRV-NETS behavior from ECA-driven architectural rules

The crucial contribution and added-value of the proposed approach to the service paradigm concerns thus the concurrent behaviour, we assign to different messages and services states. Such behavior will be clearly captured by different *transitions*, with their inherent inscriptions and conditions. For the conception of different transitions, we mainly relies on the previous detailed rule-based. That is, at this formal stage, we assume given for each business activity: (1) different informal intentional business rules, (2) their corresponding operational ECA-driven formulations and (3) highly recommended but not mandatory their disciplined ECA-driven architectural modelling.

Let us first emphasize that with respect to the formal definition of CSRV-NETS, a transition in its general pattern exhibits the following behavior. First, a transition involves some triggering events and messages, entering into contact with corresponding (parts of) service states. The result of this interaction leads to the change of invoked service states, to the absorption of the triggering events and messages and apparition of new invoked messages. Such transition behavior should only be permitted under valid constraints of the involved events / messages and service states. For this general CSRV-NETS transition behavior, it becomes how close is the gap between such transitions

and the ECA-driven paradigm. In other words, any given ECA-driven rule can be straightforwardly translated into a CSRv-NETS transition that correctly reflects its behaviour. That is:

**Event-part** : It corresponds to different triggering input inscriptions outgoing from the corresponding event place. Recalling that for each event or message type, we are associating a corresponding place to catch any event or message instances.

**Condition-part** : It has to be translated into a compatible transition condition, using declared variables for message parameters and service state properties as well as any required constants. Involved properties related to service states have to be translated into input-arc inscriptions from the service state place.

**Action-part** : It is expressed in terms of exported messages and changes targeting involved service states. That is, we have to conceive output arc-inscriptions relating the involved transition with respective (message and service-state) places associated with such outgoing messages service state properties.

► **Example 3.4.5 (The CSRv-NETS flight service behaviour)**: For the flight service interface, we have associated three transitions to reflect the (business) semantics of the three received messages, namely: The transition `Tflgh_rq` for capturing the request activity with the offered flights (i.e. `flight_requested`) as output result; the transition `Tbkfl` for capturing the booking activity and finally the transition `Tclfl` to govern the cancel activity if any.

In the following, we detail the rigorous arc-inscriptions corresponding to the transition capturing the flight-request business activity. That is, we explain how to formally derive such arc-inscriptions from the already discussed and detailed ECA-driven architectural rule for that business activity. The other transitions associated with the confirmation and cancelling activities can then afterwards be similarly derived. For them we just hint how to capture them from their ECA-driven rules. More precisely, following the above guidelines, from the detailed request ECA-driven architectural rule, the gradual construction of the corresponding transition (`Tflight_rq`) inscriptions could be summarized as follows:

- (1) To reflect the events part of this ECA-driven architectural rule, the transition `Tflgh_rq` must have one input inscription from the request message place `Flight_Request` and one from the state flight place `Flight_St`. By respecting the template message signature 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 should be:  $\langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, RsD : Dy \rangle$ , with  $R$  as the flight reference,  $Cx$  as the (normal) ticket price, and  $Dy$  the date limit for booking before losing that reservation.



- (2) To reflect the rule conditions, the transition condition should have the form:  $AvSt(Fg) - 1 \geq 0 \wedge Rs.[Cs.R] \wedge ((Cx \leq Mx) \vee ((Ag \leq 18) \wedge (Py = Cx * 0.8)))$ . That is, the available seats have 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 paid amount will be just 80 percent of the price.
- (3) Finally, as output message we have to report back the processed reservation, by considering all relevant information such as the flight references, the computed price and the date limit. This is reflected by the arc-inscription associated with the message  $FlgRqd(Cs, Fg, R, Py, Dy)$

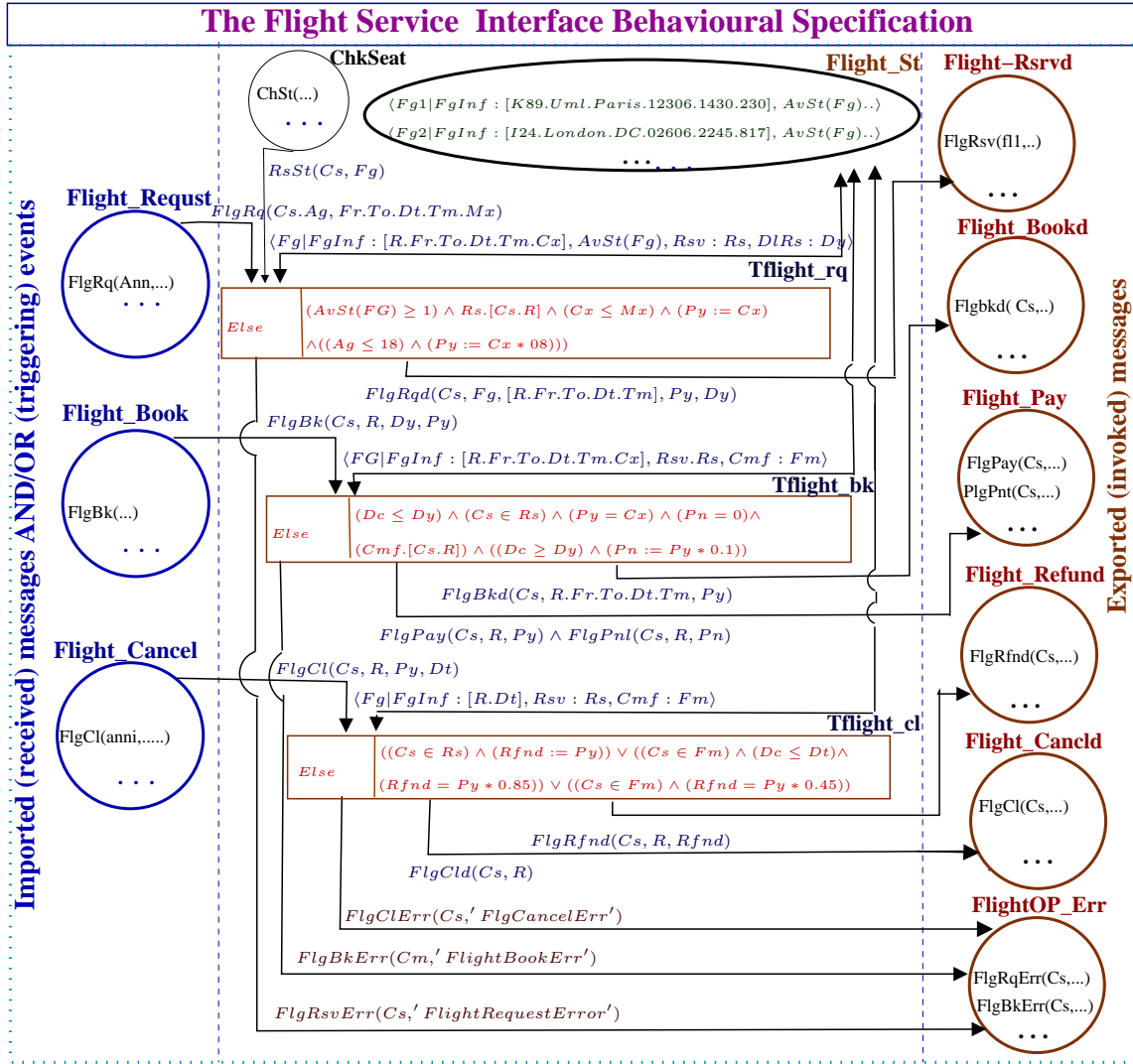


Figure 3.8: The CSRV-NETS-based Behavioural Specification of the Flight Service.

In the same spirit, we have formalized the ECA-driven architectural rules concerning the con-

firmation or cancelling activities business activities. Indeed, as depicted in Figure 3.8, we have smoothly derived the associated transitions (i.e.  $T_{bkf1}$  and  $T_{clf1}$ ) reflecting these two business activities. The confirmation ECA-driven business rule requires for instance that the reservation deadline to respected. Otherwise, a penalty is to be paid besides the ticket price.

### 3.5 CSRV-NETS: A Rewriting-logic based behavioral semantics

Re-emphasizing again that the crucial benefit, of adopting high-level service-oriented Petri nets as a formal setting for service behavior, resides in their inherent ability to provide us with executable graphical animated specification, which can further be formally validated and analyzed. The validation by rapid-prototyping at the specification level permits thus detecting and circumventing mistakes, misconception, inconsistency, etc. There are plethora of ways of semantically interpreting the behavior governing transitions (e.g. algorithmic, process-based, operational, denotational, etc.) of a given (algebraic high-level) Petri net.

In the following, we first sketch an intuitive algorithmic semantics for semantically interpreting our CSRV-NETS. This first interpretation allows us a better understanding the more rigorous concurrent operational semantics, we will subsequently and definitively adopting for CSRV-NETS, namely, a rewriting-logic [Mes92] based semantics. We should pointed out that, the forwarded rewriting-logic CSRV-NETS semantic interpretation is mainly inspired by our previous work on the CO-NETS framework for developing adaptive concurrent information systems [AS02, AS04, Aou02, ASB07]. Other work relating (high-level) Petri nets with rewriting logic include [Ste98], [BGCM94] and [BMSB93].

#### 3.5.1 An intuitive CSRV-NETS behavioral semantics

The intuitive semantics we are proposing is mainly inspired by the algebraic semantics in [Rei91]. Informally speaking, given a transition, we have to find the right term *substitutions*, which allow firing that transition. More precisely, for each input-arc inscription multi-term of a given input-place to that transition, we have to find a substitution so that the instantiated multi-term corresponds to tokens within that input place. Furthermore, such substitutions to input-arcs should result in holding the transition condition at true. When such substitutions are found, the marking of all involved (input / output) places to that transition are to be accordingly updated. That is, as usual we have to delete all consumed tokens from input places and add of the newly created tokens to the output places<sup>3</sup>.

In the following, we describe this intuitive firing of CSRV-NETS transitions, in a more understandable manner fitting the above CSRV-NETS definitions. First, we start with an intuitive representation of any CSRV-NETS transition, as a general tuple of the form:

---

<sup>3</sup>Test and inhibitor arcs do not involve any deletion or creation

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

In this tuple, `input_inscription` and `output_inscription` stand for the couple (place, multi-term). That is, for input (resp. output) places, the couple refers to any input (resp. output) place to that transition with its corresponding arc-inscription multi-term. With respect to the precise CSRV-NETS definition above, and for a given transition denoted  $t$ , this general tuple takes then the form:

$$\langle t \mid I.SvT(t), O.SvT(t), SvTC(t) \rangle$$

Furthermore, as we already detailed in definition 3.4.2, we can represent  $I.SvT(t)$  as a multiset of the form:  $\otimes_i(p_i, mt_i)$ . Similarly, the output-inscriptions  $O.SvT(t)$  are captured as  $\otimes_j(q_j, nt_j)$ . Finally we denote the condition  $SvTC(t)$  by its corresponding boolean multiterm. With these details, we finally result in the following CSRV-NETS transition firing inference rule.

**Definition 3.5.1 (CSRV-NETS-transition semantics)** We assume given a marked CSRV-NETS net, with its marking state denoted by  $\mathcal{M}_{st} = \otimes_k(p_k, M(p_k))$  as defined in definition 3.4.3. Further, we assume as above that transitions are represented as a tuple:

$$\langle t \mid \otimes_i(p_i, mt_i), \otimes_j(q_j, nt_j), [T(X)]_{bool} \rangle$$

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

$$\frac{\exists \sigma \quad x(t) \rightarrow [T_{p_i}(\emptyset)] \mid \sigma(I.SvT(t)) = \otimes_i(p_i, \sigma(mt_i)) \in \mathcal{M}_{st} \wedge (\sigma([T(\emptyset)]_{bool}) = True)}{\mathcal{M}'_{st} = \mathcal{M}_{st} - \sigma(I.SvT(t)) + \sigma(O.SvT(t))}$$

$$\text{With } \sigma(O.SvT(t)) = \otimes_j(q_j, \sigma(nt_j))$$

Note that, in this transition firing definition, we should *manually* search for the right substitution. Furthermore, it is very difficult to speak about any form of parallelism or even interleaving while firing different transitions. These are among the main severe disadvantages of such ad-hoc intuitive interpretation for CSRV-NETS transitions. In the following, we overcome that by proposing an intrinsic true-concurrent semantics based on rewriting logic [Mes92].

### 3.5.2 CSRV-NETS Rewriting-logic based semantics

Worth-mentioning is that most of Petri-net formalisms (to service foundation) adopt a direct algorithmic and hence ad-hoc behavioral interpretation, which prevents any rooms for explicitly reasoning about or evolving that behavior. We instead forward a more disciplined, declarative yet executable, efficient and flexible semantical interpretation for CSRV-NETS behavior in terms of

rewriting logic [Mes92]. More specifically, among the benefits we are targeting for such rewriting-logic (RL) based CSRV-NETS behavioral interpretation, we emphasize the followings. First, RL is based on rewriting techniques, that means on (rewrite) *rules*. This facilitates a uniform translation of any forwarded ECA-based rules governing architectural connectors. Besides that, since RL semantically subsumes logic and functional paradigms [MOM96], most of existing logic-based proposals for reasoning about (business) rules (e.g. [AA02b]) can find a common expression into RL. Second, RL is true-concurrent by essence, enhancing thus distribution and decentralization as promised by service-orientation. Third, RL is currently endowed with highly efficient MAUDE language [CDE<sup>+</sup>07], allowing millions of rewritings per-second. Fourth, with its intrinsic reflection capabilities [CM96], RL promotes separation of rules / modules specification from their compositions and executions using strategies. Last but not least, for certification purposes, RL has been endowed with property-oriented temporal logics [MOPF<sup>+</sup>05], plus the MAUDE LTL-based built-in model-checker [CDE<sup>+</sup>07].

The rest of this section is organized as follows. First, we define the notion of CSRV-NETS-*rewrite rules* that establish how any CSRV-NETS transition can be behaviorally captured as a rewrite-rule. Second, based on such CSRV-NETS-*rewrite rules*, we forward a more refined generic pattern for CSRV-NETS transitions and express it in terms of rewrite rules. We then illustrate this CSRV-NETS semantics using the Airline CSRV-NETS specification. Third, with respect to such tailored CSRV-NETS transitions pattern and rule, we define a CSRV-NETS-rewrite theory as a tailored form of the general rewrite-logic theory. Since this tailored CSRV-NETS-rewrite theory cannot be directly implemented and executed directly using the MAUDE language, we present how to leverage this language accordingly. We then apply this CSRV-NETS-compliant MAUDE leveraging, using again the Airline CSRV-NETS specification.

By adopting this logic, in the following we present a tailored instantiation, of the general rewrite theory, fitting all the so-far described (syntactical) features of CSRV-NETS. We will refer to this instantiation as CSRV-NETS-*rewrite theory*. Afterwards, we introduce the general transition pattern and its corresponding interpretation in this logic.

**CSRV-NETS rewrite theory and rules pattern.** The following definition reflects the straightforward translation of any CSRV-NETS transition behavior as a corresponding rewrite rule.

**Definition 3.5.2 (CSRV-NETS rewrite theory)** We assume given an CSRV-NETS specification following definition 3.4.2. A CSRV-NETS-rewrite theory is then a set of quadruples  $R_{sv} \subset SvT \times ([SvP \times [T_{Sv}(X)]_{\oplus} \cup [T_{Ms}(X)]_{\oplus}]_{\otimes})^2 \times (T_{Sv}(X) \cup T_{Ms}(X))_{bool}$ . The elements of  $R_{sv}$  are called rewrite rules, where for each transition  $t$  of the CSRV-NETS corresponds a rewrite rule of the form:

$$(t, ([\bigotimes_{k=1}^{n_p} (p_k, [l_k]_{\oplus})]_{\otimes}), ([\bigotimes_{k=1}^{n_q} (q_k, [r_k]_{\oplus})]_{\otimes}), SvTC(t))$$

Where:

- $[l_k]_{\oplus} \in [T_{s(p_k)}]_{\oplus}, [r_k]_{\oplus} \in [T_{s(q_k)}]_{\oplus}$  and  $t \in T$
- $TC(t) \in (T_{S_{sv}}(x(t)) \cup T_{Ms}(x(t)))_{bool}$  as a boolean expression over  $T_{S_{sv}}(x(t)) \cup T_{Ms}(x(t))$ , where  $x(t)$  denotes the variable set occurring in  $[l_k]_{\oplus}$ .
- $n_p \leq |SvP|, n_q \leq |SvP|, \otimes_k(p_k, [l_k]_{\oplus}) \in Pre(t)$  and  $\otimes_k(q_k, [r_k]_{\oplus}) \in Post(t)$

Such rewrite rules will be denoted in a rewrite rule form as usual.

$$t : (|[\otimes_{k=1}^x(p_k, [l_k]_{\oplus})]_{\otimes}|) \Rightarrow (|[\otimes_{k=1}^y(q_k, [r_k]_{\oplus})]_{\otimes}|) \quad \text{if} \quad \text{SvTC}(t)$$

**Rules for CSRV-NETS-transitions generic pattern.** We aim at automating the translation of any CSRV-NETS transition behavior into a corresponding rewrite rule, while respecting the above CSRV-NETS-rewrite theory. For that, we propose a generic pattern for CSRV-NETS-transitions, which allows capturing any rule-centric business activity in a given service-oriented business process. More precisely, as depicted in Figure 3.9, a CSRV-NETS-generic stateful and rule-centric transition, reflecting an elementary (orchestration-based) service composition, puts into contact the following service ingredients:

- (1) Events and (exported) message instances for triggering the service composition. These events and messages (i.e.  $\bigoplus_{i=1} Ms_i \wedge \dots \wedge \bigoplus_{k=1} Evn_k$ ) are initiated from a focussed-service and related to a given business activity.
- (2) The required service state parts (i.e.  $\bigoplus_{i=1}^k \langle Sid_i | prs_i \rangle$ ) are selected, to express any constraint related to the governing (business) rule of that activity.
- (3) Imported and invoked messages (i.e.  $\bigoplus_{j=1} Msi_j \wedge \dots$ ) from other services may be participating in such generic elementary composition.

By composing these service events, messages and states as inputs to a generic CSRV-NETS-transition, we expect an outcome that reflects the following effect:

- (1) The instantiated rule constraint as transition-condition being evaluated to true. That is, the participating service states with their triggering events and messages fulfil the constraints agreed on in the governing ECA-driven business rule.
- (2) The triggering events and sent message instances being consumed by associated targeted service states.
- (3) Possible changes in the involved service state-parts in accordance with the governing rule in place. That is, we result in  $\bigoplus_{j=1}^l \langle Sid'_j | prs'_j \rangle$  where the prime(') symbol reflects the changes in service state attribute values.

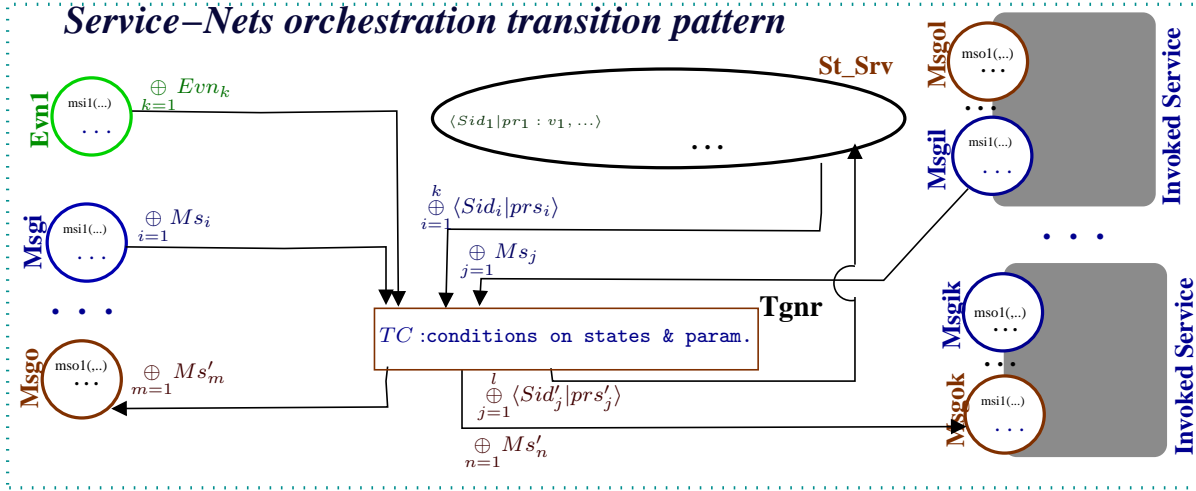


Figure 3.9: Generic pattern for CSRV-NETS-transitions

- (4) The emerging of new message instances being sent either to the main focussed-service or to other participating services. That the new messages  $\oplus_{n=1} Ms'_n$  are being emerged.

► **Example 3.5.3 (Deriving the rules of the Flight CSRV-NETS service)** By applying the afore-described general form of rewrite rules, it is not difficult to generate the transition rules associated with this CSRV-NETS account specification.

**Tflg\_rq** :  $(FLGRQ, FlgRq(Cs, Ag, Fr.To.Dt.Tm.Mx)) \otimes$   
 $(FlgSt, \langle Fg | FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs \rangle) \otimes (Rsv, RsSt(Cs, Fg))$   
 $\Rightarrow$  if  $((AvSt(FG) \geq 1) \wedge Rs.[Cs.R] \wedge (Cx \leq Mx) \wedge (Py := Cx) \wedge ((Ag \leq 18) \wedge (Py := Cx * 08)))$   
 then  $(FLGRQD, FlgRqd(Cs, Fg, [R.Fr.To.Dt.Tm], Py, Dy))$   
 else  $(FLGERR, FlgRsvErr(Cs, 'FlightRequestError'))$

**Tflg\_cfm** :  $(FLGBK, FlgBkd(Cs, R, Dy, Py)) \otimes (FlgSt, \langle Fg | FgInf : [R.Fr.To.Dt.Tm.Cx], Rsv.Rs, Cmf : Fm \rangle)$   
 $\Rightarrow$  if  $((Dc \leq Dy) \wedge (Cs \in Rs) \wedge (Py = Cx) \wedge (Pn = 0) \wedge (Cmf.[Cs.R]) \wedge ((Dc \geq Dy) \wedge (Pn := Py * 0.1)))$   
 then  $(FLGBKD, FlgBkd(Cs, R.Fr.To.Dt.Tm, Py)) \otimes (FLGPAY, FlgPay(Cs, R, Py)) \wedge FlgPnl(Cs, R, Pn)$   
 else  $(FLGERR, FlgRsvErr(Cs, 'FlightBookError'))$

**Tflg\_cl** :  $(FLGCL, FlgCl(Cs, R, Py, Dt)) \otimes (FlgSt, \langle Fg | FgInf : [R.Dt], Rsv : Rs, Cmf : Fm \rangle)$   
 $\Rightarrow$  if  $((Cs \in Rs) \wedge (Rfnd := Py)) \vee ((Cs \in Fm) \wedge (Dc \leq Dt) \wedge (Rfnd = Py * 0.85)) \vee ((Cs \in Fm) \wedge (Rfnd = Py * 0.45))$   
 then  $(FLGCLD, FlgClD(Cs, R)) \otimes FlgRfnd(Cs, R, Rfnd) \wedge FlgPnl(Cs, R, Pn)$   
 else  $(FLGERR, FlgRsvErr(Cs, 'FlightCancelError'))$

**Reasoning about CSRV-NETS services.** As a result of this specific form of rules—in which we have in particular the binary multiset operator  $\otimes$  instead of any general function  $f(x_1, \dots, x_n)$  as given in general rewrite logic entailment inferences in definition ??— the four rewriting inference rules has to be adapted in consequence. In addition, as we already pointed out, we aim to exhibit a full intra- and inter service-state concurrency. For this purpose, on the one hand we allow the service state and message instances within a given place to be split and recombined as needed; this is achieved by introducing an additional inference rule we refer to as **Service-configuration Splitting and Recombining**. On the other hand, to permit selecting from any service state just the necessary service properties invoked by given event or message, an adequate inference rule is proposed we refer to as **service-state Splitting / Merging**.

**Definition 3.5.4 (CSRV-NETS-entailment inference rules)** Given an CSRV-NETS rewrite theory  $R$ , we say that  $R$  entails a sequent  $\underline{sf} \Rightarrow \underline{sf}'$ , where  $(\underline{sf}, \underline{sf}')$  are a pair of CSRV-NETS states, **iff**  $\underline{sf} \Rightarrow \underline{sf}'$  can be obtained by finite (and concurrent) applications of the following rules of deduction.

$$(1) \text{ Reflexivity : } \forall \quad |[sf]_{\otimes}| \in ([SvP \times [T_{Stv}(X)]_{\oplus} \cup [T_{Ms}(X)]_{\oplus}]_{\otimes}),$$

$$\frac{}{|[sf]_{\otimes}| \Rightarrow |[sf]_{\otimes}|}$$

$$(2) \text{ Congruence : } \forall \quad |[sf_1]_{\otimes}|, |[sf'_1]_{\otimes}|, |[sf_2]_{\otimes}|, |[sf'_2]_{\otimes}|$$

$$\frac{|[sf_1]_{\otimes}| \Rightarrow |[sf'_1]_{\otimes}| \quad |[sf_2]_{\otimes}| \Rightarrow |[sf'_2]_{\otimes}|}{|[sf_1]_{\otimes} \otimes [sf_2]_{\otimes}| \Rightarrow |[sf'_1]_{\otimes} \otimes [sf'_2]_{\otimes}|}$$

$$(3) \text{ (Concurrent) Replacement: for each rule}$$

$$t : |[sf(x_1, \dots, x_n)]_{\otimes}| \Rightarrow |[sf'(x_1, \dots, x_n)]_{\otimes}| \quad \text{if } SvTC(\bar{x}(t)) \text{ in } R,$$

$$\frac{[sv_1] \Rightarrow [sv'_1] \dots [sv_n] \Rightarrow [sv'_n] \wedge SvTC(\bar{sv}/\bar{x}(t)) = True}{|[sf(\bar{sv} / \bar{x})]_{\otimes}| \Rightarrow |[sf'(\bar{sv}' / \bar{x})]_{\otimes}|}$$

$$(4) \text{ Transitivity : } \forall |[sf_1]_{\otimes}|, |[sf_2]_{\otimes}|, |[sf_3]_{\otimes}|$$

$$\frac{|[sf_1]_{\otimes}| \Rightarrow |[sf_2]_{\otimes}| \quad |[sf_2]_{\otimes}| \Rightarrow |[sf_3]_{\otimes}|}{|[sf_1]_{\otimes}| \Rightarrow |[sf_3]_{\otimes}|}$$

$$(5) \text{ Service-Configuration Splitting and Recombining: } \forall p \in P, \forall [n]_{\oplus} \in [T_{s(p)}]_{\oplus}, \text{ and } \forall [m_i]_{\oplus} \in [T_{s(p)}]_{\oplus}; i \in \{1, \dots, n_p\}.$$

$$\frac{|[n]_{\oplus}| = |[ \bigoplus_{k=1}^{n_p} m_i ]|}{|[ (p, [n]_{\oplus}) ]_{\otimes}| = |[ \bigotimes_{i=1}^{n_p} (p, [m_i]_{\oplus}) ]_{\otimes}|}$$

$$\frac{\forall p \in P; \forall [n]_{\oplus}, [n']_{\oplus} \in [T_{s(p)}]_{\oplus}}{|[ (p, [n]_{\oplus}) ]_{\otimes}| \otimes |[ (p, [n']_{\oplus}) ]_{\otimes}| = |[ (p, [n]_{\oplus} \oplus [n']_{\oplus}) ]_{\otimes}|}$$

- (6) **Service-state Splitting / Merging:**  $\forall \langle SvI \mid prs_1, \dots, prs_k \rangle \in [T_{Stv}(X)]_{\oplus}$ . with  $prs_i$  is a list of pairs property-values).

$$\frac{True}{\langle SvI \mid prs_1, \dots, prs_k \rangle = \bigoplus_{i=1}^k \langle SvI \mid prs_i \rangle}$$

With  $\langle SvI \mid \emptyset \rangle = SvI$

- (7) **Identity:**  $\forall p \in SvP$

$$\frac{True}{(p, \emptyset_M) = \emptyset_B}$$

### 3.6 CSRV-NETS behavioral validation: A tailored MAUDE extension

As we recalled in the appendix, the MAUDE language cannot directly support a faithful implementation of the above CSRV-NETS rewrite theory, and this at-least for the following reasons. First, whereas MAUDE supports only a holistic object-oriented perception, as a community of concurrent object state and message instances, the above CSRV-NETS rewrite theory envisions a more loosely-coupled distributed service-based configuration. Second, whereas MAUDE supports only indivisible monolithic object-state (as tuple), the above CSRV-NETS rewrite theory explicitly separate between local and observed service-states and permit their just-in-time split and recombining. Third, whereas MAUDE messaging does not distinguish between observed and local ones, we require at the CSRV-NETS an explicit distinction between events, received and invoked messages while triggering service states. Towards overcoming these serious MAUDE shortcomings, we are leveraging in consequence the MAUDE holistic object-oriented configuration. This is, towards a service-based CSRV-NETS-compliant configuration, we are re-visiting and adapting the MAUDE general configuration to cope with the following envisioned features:

**Scoped service (component) state :** To allow extraction of observed service interfaces, we propose a service component state of the form:  $\langle SvI \mid \mathbf{atl}_1 : vl_1, \dots, \mathbf{atl}_k : vl_k, \mathbf{atbs}_1 : vs_1, \dots, \mathbf{atbs}_l : vs_l \rangle$ . In this new "colorful" tuple-state, the part  $\mathbf{atl}_i$  stands for local features, whereas  $\mathbf{atbs}_j$  stands of observed service properties.

**Service State split / recombining :** Moreover, we endow that service component state with an axiomatization that permits splitting and recombining it at need. Such split / recombine axiom can be summarized as:  $\langle SvI \mid prs_1, prs_2 \rangle = \langle Id \mid prs_1 \rangle \langle SvI \mid prs_2 \rangle$  With  $prs_i$  abbreviation pairs of 'attribute:value'.

**Imported / exported service messages and events :** We further allow service messages to be observed. We thus distinguish between local service messages (to that service component)



and observed ones to participate in external service interactions. Moreover, we distinguish events as triggering messages for rules. they are messages appearing only at the left-hand side of a given rule.

Concretely, we are introducing a new MAUDE-based configuration, we refer to as service-configuration and denote by `SRVCMP_GNR` (i.e. service component generic). In contrast to the usual MAUDE object-configuration, this new MAUDE-based service-configuration is distinguished by the following features in compliance with the CSRV-NETS ones. First, we are proposing two distinct sorts `obs_StatSRV` and `loc_StatSRV`, to explicitly and separately declare observed service properties from hidden ones. Second, we separate imported / exported observed messages from local ones as well as from explicitly declared (triggering) events. Thrid, the service-state split/recombining axiom is implemented using two rules, namely `SplitAT` and `RecombAT`. Below is a sketch of the main features of such generic specification for service configurations. Note that we the split / recombining leads to two rules to be judiciously applied using the reflection capabilities of MAUDE as strategies.

```

1. mod SRVCMP_GNR is
2. subsort obs_StatSRV loc_StatSRV evnt_StatSRV < StatSRV.
3.subsort obs_StatSRV loc_StatSRV < StatSRV ...
15.op <_ | > :SRViD evnt_Prop→evnt_StatSRV .
16.op <_ | > :SRViD obs_Prop→obs_StatSRV .
17.op _ : ConfSRV ConfSRV → ConfSRV [ac] ...
21.rl [SplitAT] : < SvI|prs1,prs2 >⇒< SvI|prs1 >< SvI|prs2 > .
22.rl [RecombAT]: < SvI|prs1 >< I|prs2 >⇒< SvI|prs1,prs2 > .

```

► **Example 3.6.1** Using the MAUDE workstation environment<sup>4</sup>, Figure 3.10 depicts the concrete and complete implementation, of both the structural and behavioral features of the Flight service, we discussed and detailed in the previous sections. Important to notice is the importing of the service-configuration; otherwise the specification cannot be performed. After de specification of all involved sorts for messages and service states, first the Flight service state properties are defined. Then, the involved events and messages are declared. Equations for defining the requested discounts are then specified. Finally, after defining all needed variables, we formalize the three rules reflecting the request, confirmation and canceling business activities.

To validate this compliant CSRV-NETS-Flight MAUDE-based operational service specification, we have experimented it through several concrete scenarios. Figure 3.11 depicts a simplified snapshots from these experimentation. That is, we assume having some flight state instances and involved triggering events and messages for booking, confirming and cancelling such flight instances. After running such service flight configuration, we result in new final configuration as depicted in Figure 3.12, where all messages have been consumed and the flight states have accordingly updated. In

<sup>4</sup><http://moment.dsic.upv.es/>



```

FlightConf3.maude*
mod Flight-Conf is
ex Flight-Service .

op AirFrance : -> Flgld .
op KLM : -> Flgld .
op Lufthabsa : -> Flgld .
op Hainan-Airlines : -> Flgld .
ops CS1 CS2 : -> Custld .
op AG1 AG2 : -> AGCYld .
ops SCF SCFRq SCFBk SCFCI : -> SvConf .

eq SCF = FlgRq(CS1, ["Nas" # "Magdeburg" # 1 # 0], AG1, ["001" : "Berlin" : "Paris" : [30, 3, 2009] ; 550])
  < AirFrance | FlgInf: ["AF932" . "Berlin" . "Paris" . [30, 3, 2009] . 500], AvSt: 15, RsvP: nul, CfmP: <CS2, "2", 1>, DIRs: [28, 3, 2009] >
  FlgBk("1", CS1, AG1, "AF932", 500, [14, 3, 2009]) FlgCI(Cs1, AG1, "AF932", 500, [20, 3, 2009]) |
endm

```

Figure 3.11: Concrete CSRv-NETS-Flight service configuration scenarios

reality as will be detailed in the fourth chapter, we have explicitly controlled the execution through reflection strategies to avoid looping or canceling before flight requesting and similar inconsistent behavior.

Emulator	Result
	<pre> Maude&gt; reduce in Flight_Str : downTerm(Compute(upTerm(SCF)), 'error) . reduce in Flight_Str : downTerm(Compute(upTerm(SCF)), 'error) . rewrites: 81 in 609120347ms cpu (16ms real) (0 rewrites/second) result SvConf: FlgRqd(CS1, ["AF932" . "Berlin" . "Paris" . [30, 3, 2009] . 500], AG1, 500, [28, 3, 2009])   FlgBkd(CS1, AG1, ["AF932" . "Berlin" . "Paris" . [30, 3, 2009] . 500]) PayFlg(CS1, AG1, "001", 500)   FlgCI(CS1, AG1, "AF932", 425) &lt; AirFrance   FlgInf: ["AF932" . "Berlin" . "Paris" . [30, 3, 2009] . 500],   AvSt: 15, RsvP: nul, CfmP: &lt;CS2, "2", 1&gt;, DIRs: [28, 3, 2009] &gt; . </pre>

Figure 3.12: The resulting CSRv-NETS-Flight configuration by running the previous one

## 3.7 Chapter Summary

In this chapter we put forwards the milestones towards an advanced disciplined and stepwise approach for developing adaptive service-oriented applications. The approach capitalizes on advanced software-engineering concepts and mechanisms including: Business rules and stereotyped UML-classes, tailored high-level service-oriented Petri nets, rewriting logic and its efficient MAUDE language and ECA-driven architectural interactions. The approach is illustrated and validated through a typical rule-centric flight service. The chapter concentrates on the shifting from the informal to the formal decisive phase. That is, we first presented how service requirements can be captured through stereotyped UML-class diagrams and event-driven business rules. We then put forwards a service-oriented Petri nets framework, that directly built on this intuitive phase. The

framework has been semantically governed by a tailored rewrite theory interpreted in rewriting logic, and implemented by accordingly extending the MAUDE language.

## Chapter 4

# Collaborative Services—Choreography meets Orchestration

In contrast to the monolithic and global component-based composition[Gri98], the service paradigm provides the developer with at-least two distinguished visions for composing services. As we reported in the second chapter, towards building further added-value realistic service-oriented business processes and applications, the service paradigm and particularly its technology is endowed with the so-called service *orchestration* and service *choreography* for composing services.

Orchestration-based service composition is inherently associated with the BPEL standard [CGK<sup>+</sup>04]. Therefore, the orchestration primarily focusses on composing a *new service* from existing services. It provides a common pattern of describing the process-based behavior of how different services need to work together, mostly by exchanging messages, in order to realize another service. As such the orchestration is based on a single-service perspective. That is, the orchestration always represents control from *one partys* perspective, as it enables the definition and execution of business processes, with respect to a specific reference Web-Service. For instance, when we tackled the flight service, though other services such customers and banks have been involved, the main focus remained on the specification of that flight service (with the support of customer and bank services).

Orchestration-based service composition as being defined cannot thus be used to describe a system of services as peers, since it yields a new service. This service-focussed vision represents one of the main difference to the choreographical composition. We should further point out the (BPEL-based) orchestration has been attracting more attention and investigations, both at the academia and industry, when compared to the choreography. This orchestration-trend is mainly due to the availability of variety of advanced BPEL-driven engines (e.g. BPEL4WS, BPEL4J), among other technological and business reasons. For instance, the intrinsic decentralization of

composition control promoted by choreography still remains a difficult and challenging problem. Moreover, business ownership and other legal issues empower the single-partner control over shared choreography-based responsibilities.

Choreography instead promotes more balanced collaborative and decentralized *multi-party* service composition. It allows each involved party to explicitly describe its role in the interaction. Choreography mainly tracks the message sequences to be exchanged among multiple parties (i.e. Web-Services), rather than focussing on a specific business process that a single party executes. Web Service Choreography relates to describing externally observable interactions, in a system composed of several web services. WS-CDL [KOMC04] is the current standardized language for describing such choreographical multi-party contracts, across a number of services (more than one). Indeed, while WSDL describes web services interfaces, WS-CDL describes collaborations between web services. WS-CDL is primarily for the case where multiple parties (business partners) do not want to expose their actual business processes to each other. Its purpose is to clearly define the interoperability needed to realize *a system* composed of many services.

In this sense, a system build of several services and described in WS-CDL, may be effectively realized as a set of peered islands of (BPEL-based) orchestrations. That is, based on the rules of engagement set by the choreography logic, such collaborating services have to work together to achieve the expected system's overall behavior. From this observation, it comes the crucial *complementarity* of the choreography and orchestration while composing realistic services. Unfortunately, this potential choreography-orchestration synergy remains badly unexplored, through the disadvantaging of the choreography vision at the expense of the service-focussed orchestration. Indeed, though the benefits of such complementary have been stressed at the descriptive-level [Pel03, BDO05], only few proposals have rigorously addressed the problem, mainly by adopting process algebras-based foundation [BGG<sup>+</sup>05, vdAWMO<sup>+</sup>06, MDM09]. Such investigations have been specifically tackling the (un)coherence while projecting the system's choreographical message exchanges on respective orchestrated local services and their messages. At the technological level, we should mention the so-called BPEL4Chor language [DKL<sup>+</sup>08, DW07]: A process-centric extension to BPEL towards coping with choreography besides orchestration.

The purpose of this chapter aims thus at promoting the potentials of this choreography-orchestration or "system-service" complementarity, while going beyond the elementary studies of local-global message exchanges. More specifically, capitalizing on the resulting stepwise approach from the previous chapter, we propose to leverage the forwarded CSRV-NETS-based orchestration towards a consistent yet behavioral and rule-centric choreography, where systems of CSRV-NETS-based services can be developed. The remaining sections of this chapter are structured as follows. In the next section, we bring more motivation and insights on how to leverage the proposed approach, from independent CSRV-NETS-based orchestrated services toward complex system of choreographically collaborating composite services. The third section delves into the first step towards such leveraging, namely the intuitive conceptualization of a tailored generic pattern for cross-organizational

inter-service event-driven business rules. In the third main section, we progressively formalize and illustrate such harmonious ECA-driven complementarity between the orchestration and choreography, by soundly extending the CSRv-NETS framework. Towards graphically attracting the reader's attention, we slightly "re-paint" the general approach architecture from the previous chapter (i.e. Figure 3.1). More precisely, in this slightly re-painted Figure 4.1, we have first put into background the non-concerned steps by this chapter. Second, at the concerned middle phase, we replaced the single CSRv-NETS-based service by supposedly "choreographically" collaborating services.

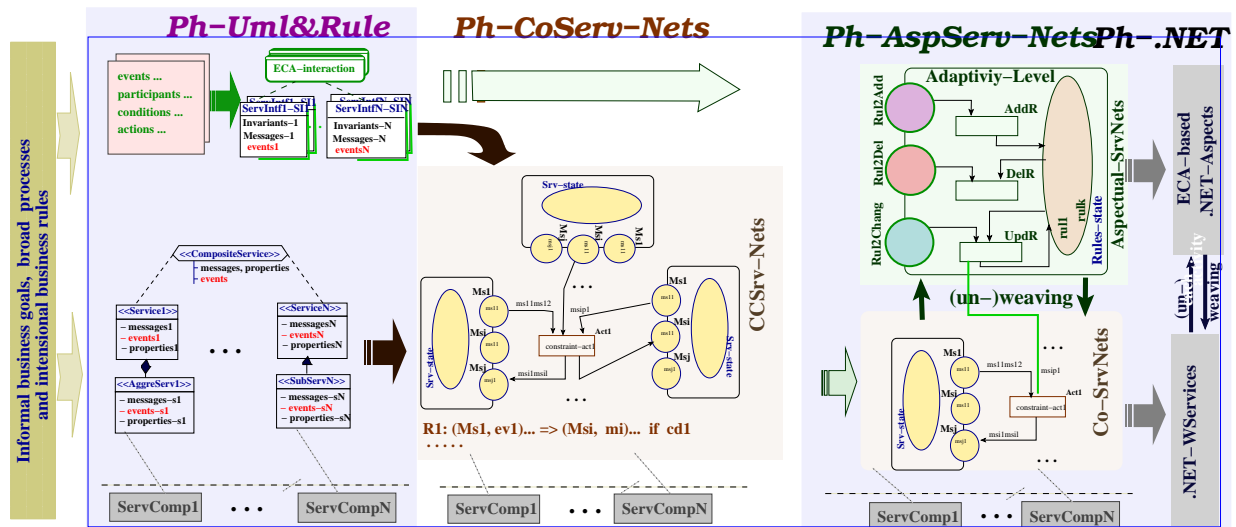


Figure 4.1: The Choreography with CSRv-NETS as third phase in the forwarded approach

## 4.1 Choreographical Services Composition with CSRv-NETS: Further Motivations

In the previous chapter, we demonstrated how service interfaces and elementary services can be progressively and rigorously leveraged to cope with concurrent rule-intensive behavioral features. We achieved that, first by informally describing service structural features using stereo-typed UML class-diagrams while capturing their behavioral features as architectural event-driven business rules. We then proposed a rigorous formalization and certification through a tailored service-oriented high-level Petri nets framework, we refer to as the CSRv-NETS. CSRv-NETS permit thus for precisely defining structural features with explicit service states, and then smoothly capturing behavioral features governed by a true-concurrent rewriting-logic semantics. This important achievement encouraged us towards pushing the scope of this approach one step further, so that complex *service systems* as collaborating *composite* services can be harmoniously tackled on top of already specified and validated individual CSRv-NETS-based services.

More precisely, as we just discussed in terms of Web-Services terminology, in the previous chap-

ter we presented how individual services (e.g. Flights, Hotels, banking, etc.) can be *orchestrated* in a behavioral, conversational and rigorous manner. In this chapter, we instead concentrate on the *choreographical* composition of several participating (elementary or complex) services to develop complex systems as *cooperating* services. A typical illustration is the travel-agency system that requires the composition and coordination of several independent services such as: Airlines (and/or trains/taxis/renting cars), Accommodations (hotel/hostel/private), financial institutions (bank/credit-card), attraction services, etc. Each of these participating services are themselves very complex, conversational, stateful and adaptive. We re-emphasize that WS-CDL [KOMC04], as standard for choreography, proposes only pure static and structural descriptions, and does not scale up to complex composite decentralized services. Furthermore, WS-CDL neither tackles behavioral, adaptive and / or concurrent features characterizing today's complex services, nor provides any founded means for specification and certification.

Besides these serious deficiencies, existing proposals to Web-Services explicitly *differentiate* between orchestration and choreography, and focus mostly on BPEL-like orchestration than WS-CDL-related descriptions. Indeed, choreography is surprisingly neglected with the strong believe that BPEL-like descriptions of (elementary and composite) services seem quite satisfactory in most cases. Unfortunately, with the current limited static and structural characterizations of Web standards (e.g. BPEL and WS-CDL), the striking necessity of choreography in *complementing* orchestrated descriptions and vis-versa, have just been further confused and obscured.

One of the main purpose of this section consists thus in demonstrating that, when the emphasize is put on (rule-centric) *behavioral* 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-oriented design and thereafter non-conform service deployment. To cater for such essential complementary, we propose to optimally benefit from both service orchestration and choreography through a harmonious synergy. Before delving into the formalization details about how to choreographically composing services, as a sound extension to the presented behavior-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-oriented applications.

- (1) By opting for an ECA-driven approach, we are in fact assuming that both independent services as well as their *collaboration* are governed by suitable event-driven *business rules*. That is, as we so-far did, *intra-service* business rules allow governing independent service behaviors, and thus exclusively focus on a single service at-hand. For instance, while specifying the Flight 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-service* business rules should govern composite services involving different regulations, policies and strategies for *correctly collaborating* involved services towards a complete system of services. In this sense, the Travel-Agency



composite service, for instance, has to be governed by respective policies and regulations for optimally and legally collaborating different involved (elementary) services such as hotels, airlines, banks, attractions, etc.

- (2) Having explained the relevance of such two-level categories of business rules—namely those for governing intra-service behavior and those for cooperating such services—the remaining central question consists in: *How to establish a behavioral relationship between such intra- and inter-service business rules while composing services?*

Towards answering such crucial question, in Figure 4.2 we are graphically forwarding an intrinsic vision for such relationship, to coordinate independently formalized orchestrated CSRv-NETS-based services. More precisely, the general conceptual milestones, we are forwarding towards a harmonious and rule-centric behavioral complementarity between service-level orchestration and system-level choreography, can be explained and motivated as follows:

- 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 intra-service business rules of different independent services (e.g. Flights, Banks and Hotels). Due to the autonomy of such services from each other, at the orchestration level, we could not report on how requested / provided (events and) messages in such services 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 under which circumstances.
- With the explicit conceptualization of the choreographical-level over orchestrated services, 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 fulfill the expected rule-based choreographical global behavior of the system. It also becomes further meaningful, why we are labeling the resulting outgoing / provided messages from such involved services as *behaviorally-certified messages*.
- As depicted in the Figure, by collaborating more than one service in a composite choreography, different messages have to be accordingly requested by such composite level from the participating services. Once such messages are received by the corresponding independent services, 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 behaviorally-certified outgoing messages or result in non-conform incorrect exception messages. Only behaviorally-certified (through service interfaces orchestration) messages can further be invoked at the choreographical composition level, to realize the expected value-added collaborative business activities with respect to the inter-service business process at hand. In other words, there is now a

necessity and benefits for such two-level complementarity, which could not be discerned with just structural capabilities of Web standards like BPEL and WS-CDL.

- 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.

#### 4.1.1 Choreographical composition within the Travel-agency

Concretely with the vacation running case study, the composite Travel-Agency once receiving a triggering request from the customer, it first validate it with respect the customer service rules (i.e. checking conditions such as age, address, minimal budget, name, etc.). Then depending on the general agencies regulation and policy rules for specific context-aware incentives, the agency dispatches different request messages to involved respective services to get Tickets, Accommodations, etc. In contrast to usual (BPEL or WS-CDL) Web-standards, with the forwarded two-level service approach, there is *no assumed systematic implicit "positive" replies* from such invoked service interfaces, even when being fully available. That is, to get Tickets, Accommodations, etc, different current local behavioral business rules put into place have to be checked and validated. This is, only in the positive case, the Agency can proceed further with the vacation business process (with the check-out of bank in the same way, etc) .

Towards a conceptualization of this rule-based choreographical behavior for composing services, we first propose a general pattern for required cross-organizational choreography-level business rules. Then, we formally extend the CSR-V-NETS framework to rigorously reflect such interaction-driven business rules, which are more close to *transient architectural connectors* [PPSGS04]. Finally, we illustrate this crucial behavioral conceptualization of choreography with the running travel agency.

## 4.2 Business-Rules pattern for Behavioral Choreography

To capture cross-organizational composite business rules, unlike intra-service business rules, we have to take into consideration besides the concerned 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 (behaviorally 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

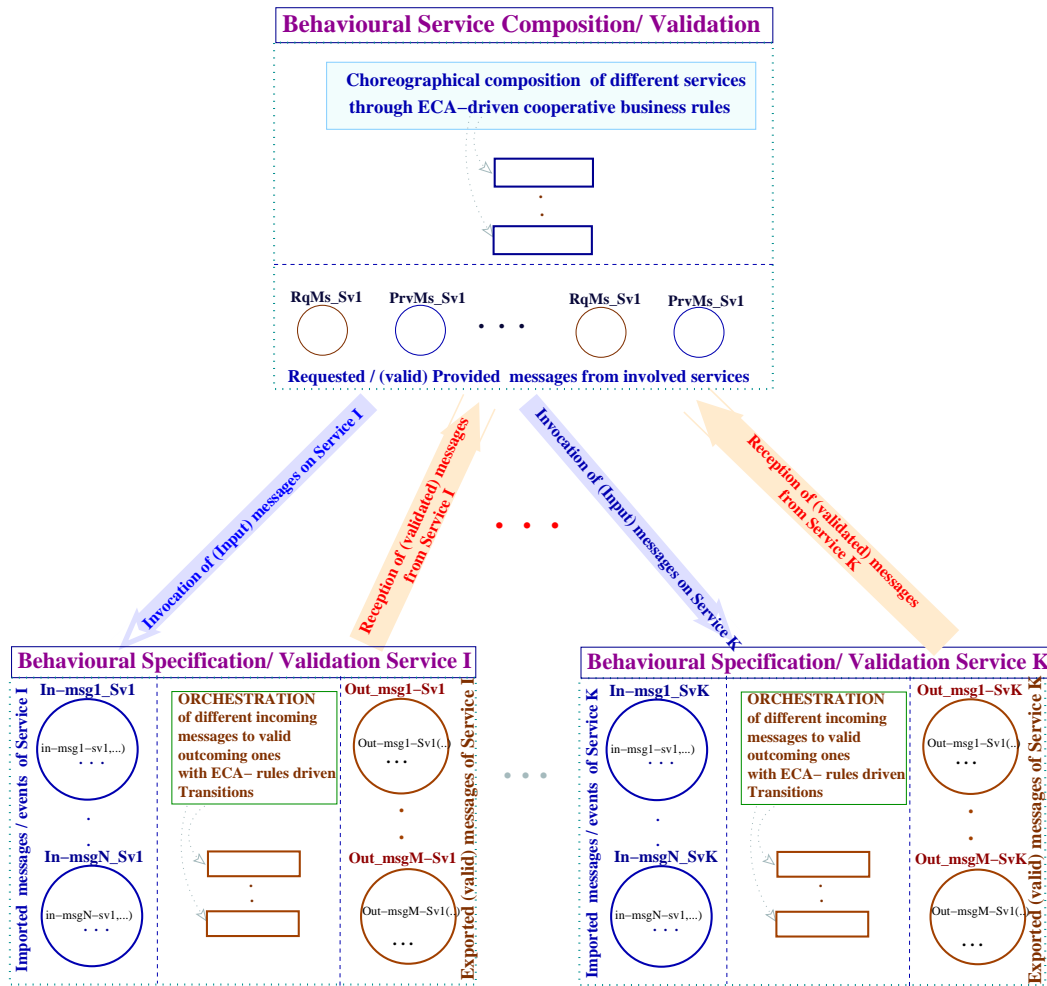


Figure 4.2: An Illustrative Complementarity of Orchestration and Choreography in the CSR-V-NETS Approach .

formulating any inter-service choreographical business rules. Besides that, specific properties from participating states, such as identities and other properties, may be part in the composition. We further note that, in contrast to intra-service rules where single service interface can be involved, at the choreography-level we must have at-least two participating services (apart from the trivial customer service). Otherwise, we could not speak about collaborating system of services.

**Extra proper properties for interacting** : Depending on the composition behavioral semantics, additional extra composition-driven properties such as messages, (stateful) attributes are to be declared for expressing the intended composition. Such information can be provided as an extra-service aimed at coordinating the other involved services (e.g. the travel-agency).

**ECA-like effects on collaborating 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. Such conditions are 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. As will be detailed below, as intra-service business rules come also into play, we suggest to explicitly declare at the composition-level the events for triggering such rules. That is, we distinguish between events triggering (choreographical) inter-service rules from those triggering (orchestrated) intra-service rules.

Towards expressing this behavior-driven choreography in a disciplined but still at a descriptive-level, while reflecting these constituents, we propose a general pattern for such cross-organizational behavioral business rules as event-driven architectural connectors. This general pattern respects the following form.

**Choreographical ECA-behavior** *<Service-Composition-Identifier>*  
**participant interfaces** *<list-of-service interfaces>*  
**invariants** *<possible extra-interaction constraints>*  
**attributes/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>*

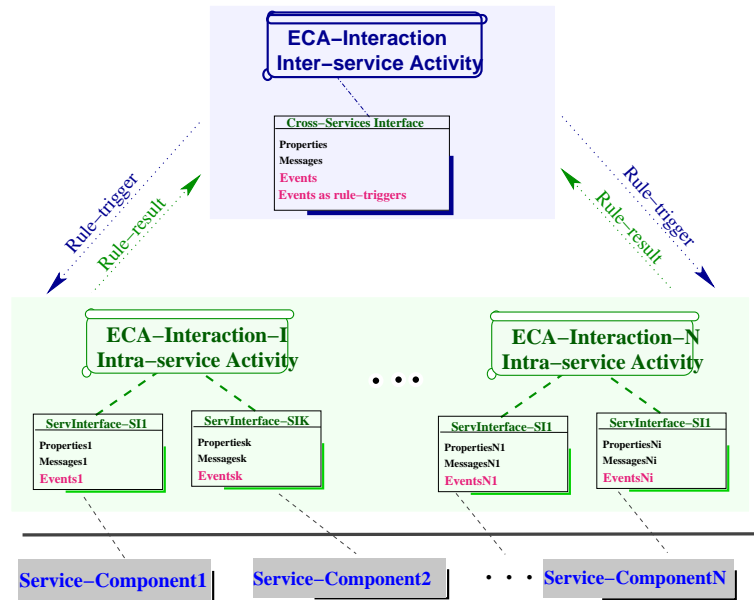


Figure 4.3: The general pattern of cross-service (choreographical) ECA-driven business rules

As depicted in Figure 4.3, the choreographical cross-service rules bring into play the involved intra-service rules to behaviorally harmonize and validate the collaborating services. Furthermore,

as we emphasized, important in this behavior-driven services composition is above all the participating service interfaces. Secondly, when required we have to specify additional invariants and constraints to be observed during the collaboration. Thirdly, besides exchanged messages, stateful data and events from the participants, we may need additional properties such constants, attributes and messages at the composition level to express complex interaction patterns. 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 directly initiate other semantically related composite rules. This behavioral 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 to be already been specified and certified (i.e. orchestrated) in order to take part in a given choreographically-driven cross-service business rules.

#### 4.2.1 Cross-service business rules for 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-organizational business rules regulating the cooperate behavioral functioning of travel agencies, with respect to participating services such as: Accommodations (hotels, hostels, apartments, etc), Transportation (e.g. Airlines, Trains, Renting-Cars, etc), Distraction (Visiting sights, attraction places) and Financing (Credit-Cards, Banking, etc), among other participants. An illustrative 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 reduction percent, we denote by P with  $10 \geq P \leq 30$  as illustration. When they decide for specific accommodations 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-organizational rule in compliance with the above general rule pattern, we further require several specific information from the composing agency-travel service. First, besides its name and address, an agency should have usually favored airline partners, favored accommodation hotels (series), and not least favored destination locations with corresponding basic prices for specific periods (for individual customer). For sake of exhibiting more behaviors, 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 favors. Taking this agency

knowledge into play as well as the capabilities of the three involved services (e.g. airlines, accommodations, banks but 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 triggering event **Request\_Travel** from the customer service, has to check whether the destination belongs the privileged ones, and more importantly if the "spending-threshold" amount set by the customer is within the range of the expected total budget. In the affirmative case, the agency submits a corresponding request like **Find\_Travel**. This composite request message should involve different request messages for ticket, accommodation and others involved services (e.g. attraction visits, car renting, 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. airlines-ticket booking, accommodation booking, etc), the agency asks the customer service to confirm or infirm the composite complete offer, which has also 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 rules 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, he/she must pay in this case an accordingly rule-driven penalty amount.

#### Choreographical ECA-behavior `Travel_Agency`

##### participant services

```
Flg: Airlines
Acom: Accommodation
Cust: Customer
Bank: Bank
```

##### constants

```
Prc1, Prc2: [0..1]
```

##### attributes

```
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_Penalty
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.Flq_Reqst(CsInf, FlgInf, 3/4*Prc1*CxT) and
    Acom.Hotel_Reqst(CsInf, HtlInfo, 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.Flq_Reqst(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(FlgRefi), Htl_Rsvd(HtlRefj))

interaction rule: Confirmed_Trip (sub-rule3)
at-trigger Cust.Trip_Confrimd(CsI, FlgRefi,HtlRefj)
    under (True)
    acting Cust.Trip2Book(Flg2Bk(CsI, FlgRefi), Htl2Bk(CsI, HtlRefj))

interaction rule : Booked_Trip (sub-rule4)
at-trigger Cust.Trip_Booked(CsI, FlgRefi,HtlRefj)
    under (True)
    acting Cust.Trip2Pay(Flg2Bk(CsI, FlgRefi), Htl2Bk(CsI, HtlRefj), Cost)

end Std-withdraw interaction rule: Cancel_Trip (sub-rule5)
at-trigger Cust.Trip_Cancel(CsI, FlgRefi,HtlRefj)
    under Trip_Booked(CsI, FlgRefi,HtlRefj) = True
    acting TripCancelled(CsI, TrRefi)and Cust.Penalty2Pay(CsI, TrRefi, Cost-Penalty(formulas))

end Std-withdraw interaction rule: Pay_Trip (sub-rule4)
at-trigger Cust.Trip_Pay(CsI, TrpRefj, Cost(Trip))
    under Trip_Booked(CsI, FlgRefi,HtlRefj) = True
    acting Bank.TripPaid(CsI, Cost(Trip))

end TravelAgent_Rules

```

As depicted on the top of this choreography-driven cross-service business rule, four partners are involved namely the Airlines, accommodations, 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 instance of the bank service.

As we emphasized, usually within each Travel agency there are seasonal offers such a list of privileged trips with attractive prices, list of selected accommodations and /or list of privileged flights and airlines partners. Each element in these list is composed with all necessary information; so, for instance, for a given privileged Trip one may found the destination, a range of costs depending of the duration and the date limit of such offers. The same is to observe for the privileged flights and accommodations. We have regrouped all these specific information as attributes at the composite service level. In addition to these variables, to capture the different discount percents, we have defined some constants (e.g. `Prc1`, `Prc2`) those values ranging over the interval  $[0..1]$ , 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 manner, 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 we motivated we propose to proceed in a conversational process-centric way, where the behavior of each involved business activity is captured through associate cross-service business rule. These business activities consist in: (1) Looking for candidate Trips (e.g. associated flights, accommodations, etc); (2) choosing the best Trip from the found candidates; (3) Confirm the selected Trip ; (4) Paying that Trip or Canceling 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 the customer service, with information about this requester, the wished trip details as well as the maximal cost that can be invested in such a Trip. In this context a "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.) through its CSRV-NETS specification. 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 is being recognized 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 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 formulas of possible discounts so that we set the maximal fares to be enforced while requesting candidate flights and accommodations (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 accommodations should in consequence not more than the  $1/4 * Prc1$  of that price.

In the same spirit of such modelling, one can further define any possible business rule 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 requesting just the flight, instead of a complete Trip (i.e. no accommodation), a similar discount formulas has been suggested. For instance, by requesting for flights before two weeks before from traveling, then a discount of  $Prc2$  is to be observed. In the same 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`.

### 4.3 Leveraging CSRV-NETS to ECA-driven Behavioral Choreography

After motivating and intuitively presenting how composing services in cross-organizational alliances, on the basis of inter-service ECA-driven business rules, and this once individual involved service interfaces have been specified and validated. The purpose of the following sections is to leverage these intuitive descriptions to a more *rigorous level*. As we pointed out, we aim achieving that by soundly extending the sofar 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 formalize the behavioral features. Finally, we 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-organizational business rules to this new choreography-driven formalism.

#### 4.3.1 Structural features in CCSRV-NETS

As explained above, to achieve a conversational and behavior-driven composition we require in most cases *proper* properties, besides requested / received messages from the participating services. These proper properties may be attributes and/or (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 setting for formally specifying them. That is, all proper attributes

are gathered in a tuple form, we refer to as the composition service 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 primitive 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 unified message, let us consider again the the Travel-Agency case with in mind the choreography general form we depicted in Figure 4.2. We clearly observe that typical agency activities such as "Request-Trip", "Trip\_confirm" and/or "Trip-Select" all consist in accordingly (i.e. in conformance with corresponding business rules) *dispatching* or sending requesting messages to corresponding participants (e.g. airlines, accommodations, attractions, car-renting, 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. Similarly, unified messages apply to `Select4Trip`, etc.

In terms of algebraic concepts, we propose to declare such unified message types as **super-sorts** of respective detailed service-related message. 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. The benefit is that 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 depends now directly on the customer wishes. For a customer requesting just flight, the composite message `Request4Trip` will include only the `request4flight` message. We are thus dynamically reshaping the composite message depending on the participant services. At this syntactical structural level, we use the symbolic notation  $\ll$  instead of the usual subset symbol  $<$  to distinguish this notion of composite message. At the behavioral Petri Nets composition level, there are manifold benefits of this notion of unified messages as we detail later.

### The Agency Structural aspects in CCSRV-NETS

Following the above clarifications and the detailed cross-service business rules for the composite Agency service, we present below the corresponding CCSRV-NETS structural specification. In this algebraic specification for composite services to enhance more the understandability we are introducing new primitives (keywords) such as: **Participants** instead of **including** and **super-sort**  $\ll$  instead of the usual subsort  $<$ . Besides that, instead of the starting keyword "obj", we are using the new expressive keyword "**Composite-Service**". Important to mention here is that we assume as

already specified all required Data, we gather in a data-level datatype we refer to "AGENCY-Data".

```

Composite-Service Agency-Service is
  extending Service-state
  protecting AGENCY-Data.
  participants Customer, Airline, Accommodation, Bank < Service .
  supersort Trip_ToFind >> Flg_Reqest 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_Confirm : 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 .
    fgi : FlgRef ; hti : HotRef ;
    Dt, DepDt, Now : Date
    MxC, Cxt, CsTT : Money

```

**Composite-Service.**

### 4.3.2 Behaviorally composing services with CCSRV-NETS

With CSRV-NETS capabilities in capturing stateful service interfaces, we present in the following how choreographical ECA-driven behavioral rules enhance these potentials towards more adaptivity in complex services. Following the same intuitive guidelines for constructing CSRV-NETS service interfaces behavior from informal service applications, the modelling steps for integrating such choreographical architectural behavior 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 component specification by algebraically specifying different properties (states, messages, events, etc.). Secondly, by gathering different composite service attributes and participants into states, we then associate it a type and a corresponding *place*. Similarly, 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 supersort symbol  $\gg$  as described above) we associate a *fusion* place (as defined in Coloured Petri Nets for instance [Jen92]). 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).

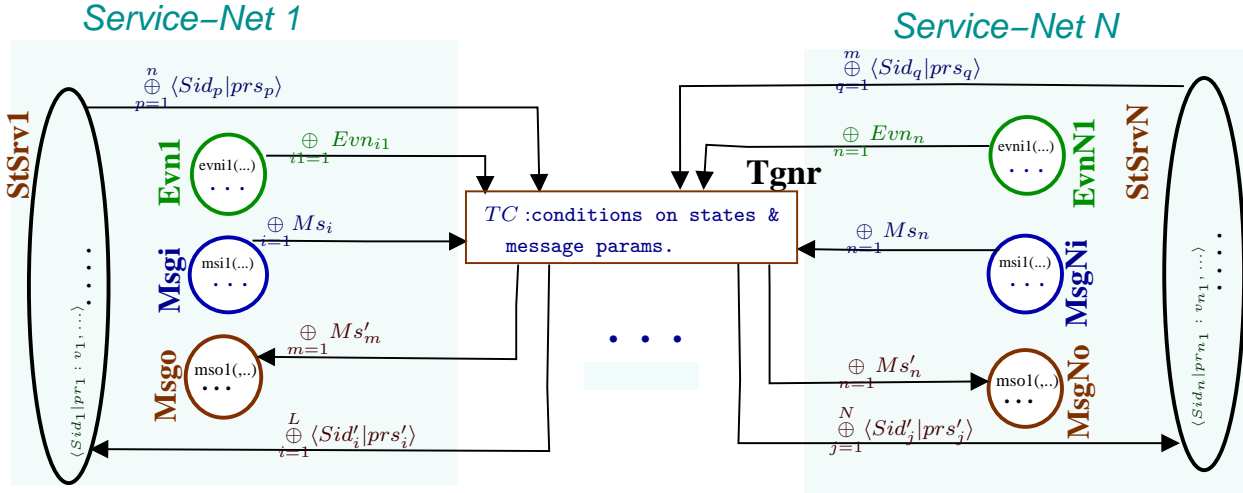


Figure 4.4: The transition pattern for collaborating services within CCSRV-NETS

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

**Definition 4.3.1 (CCSRV-NETS behavioral choreography)** Assuming that  $k$  CSRV-NETS (basic) service interface specifications are participating in a composite service, we then define  $(\prec S_{D_i} \cup \{S_{S_i}\} \cup S_{Ms_{l_i}} \cup S_{Ms_{o_i}}, \{Op\}_{S_{S_i}} \cup Ms_{l_i} \cup Ms_{o_i} \succ, \text{CSRV-NETS}_i), i \in \{1, \dots, k\}$ . We define a template community as a pair of the form:  $(\prec \bigcup_{i=1, \dots, k} S_{D_i} \cup \bigcup_{i=1, \dots, k} Bs(S_{S_i}) \cup \bigcup_{i=1, \dots, k} S_{Ms_{o_i}}, \bigcup_{i=1, \dots, k} Bs(Op_{S_{S_i}}) \cup \bigcup_{i=1, \dots, k} Ms_{o_i} \succ, \text{CSRV-NETS}$ , where:

- $\bigcup_{i=1, \dots, k} S_{D_i}$  is a union of all sets of data sorts.  $\bigcup_{i=1, \dots, k} S_{Ms_{o_i}}$  are the (message) sorts corresponding to imported / exported messages;
- the 'choreography' service net, CCSRV-NETS, reflects the behavioral composition of different CSRV-NETS services, those transitions should respect the general pattern given in Figure 4.4.

Please not that the forwarded rewrite theory for CSRV-NETS, we detailed in the previous chapter remains fully applicable to this extended CCSRV-NETS variant. All what we further require is to accept all rewriting rules associated with the transition's general pattern depicted in Figure 4.4, for collaborating different existing CSRV-NETS-based services.

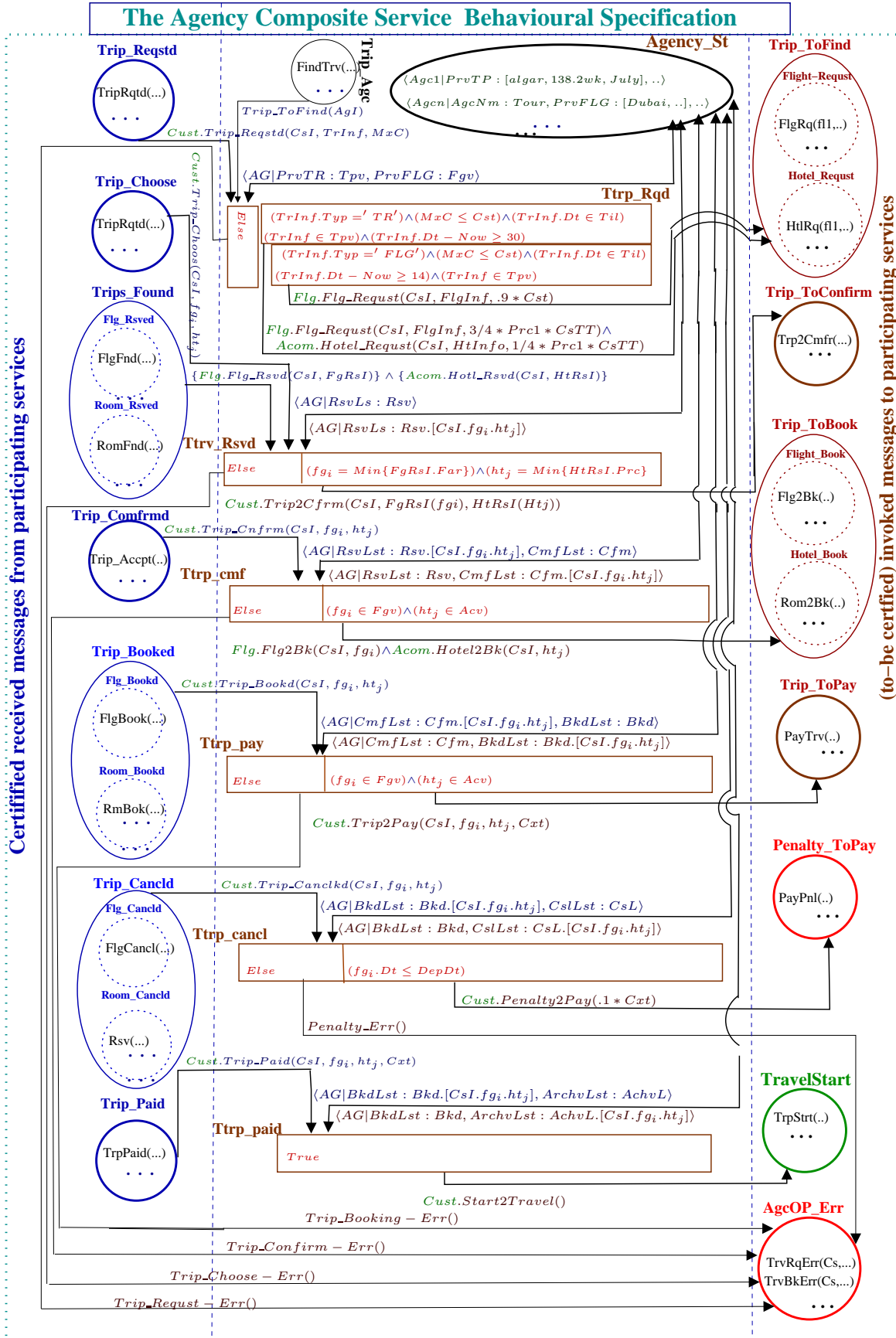


Figure 4.5: Behavioral Choreographical Specification of Travel Agency Service

## 4.4 CCSRV-NETS-based Formalization of the composite 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 4.5.

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 (i.e. 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 `Trip_Find`, we are deriving the transition `Ttrp_Rqd`. The condition of this transition reflects exactly different conditions from that business rule. The input arc-inscriptions select all the required messages and service state parts from the place `Agency_St`. For the transition `Ttrp_Rqd`, the first alternative concerns the request for just a flight (i.e. `(TrInf.Type = "TR")`). 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-service 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 assumed being certified, to respective intra-service business rules.

## 4.5 Chapter Summary

Besides service-focussed orchestration service composition, the service paradigm provides a complementary higher-level system-focussed composition. Despite the promising potentials of such system-focussed composition, in mastering more complex and adaptive scaled service-oriented applications, it remains still much work to do. We particularly contributed to the complementarity of this orchestration-choreography service composition, from a behavioral rule-centric point of view. Indeed, most of existing explorations have been addressing such complementarity only from an exchange of messages vision. We further presented how cross-service business rules should be conceived, in conformance with respective intra-service rules. The chapter illustrated these concepts through the travel-agency case-study.

## Chapter 5

# From Design- to Runtime adaptive services—Foundation and Deployment

Most of today's software-intensive applications are required to exhibit high-level of agility and adaptability, as they are operating in so volatile and competitive ('socio-techno-economical') ever-growing environment. Moreover, due to the fact that most of modern software are being dependable and mission-critical, such adaptability has to be inherently dynamic and without stopping or decreasing the capabilities of the running system. These urgent requirements towards runtime adaptability become more acute when it comes to distributed enterprise information systems, as main trigger for emerging the service technology. Indeed, as such systems are aimed at (semi-)automating target (cross-)organizational realities, they have to inherently mirroring most organization's peculiarities. In particular, due to market volatility and globalization, organizations are forced to offer just-in-time solutions, tailored to the needs of very demanding users and customers. Moreover, market pressure and advances in computation and wireless communication, have been urging organizations to shift from centralized standing-alone companies towards more loosely-coupled networked agile cross-organizational alliances, where *dynamic* evolving business interactions are the steering forces.

Abstracting from dynamic adaptability, we should again re-emphasize that the service technology and its standards represent nowadays the best emerging technological innovations, towards faithfully (semi-)automating such agile opportunistic inter-organizational alliances. Indeed, this technology treats distribution, interaction, loose-coupling and heterogeneity as main driving principles. Web-services (WS), as main enabling of the service-oriented architecture (SOA), are platform-independent self-contained software entities, with explicit interfaces. Web-Services are adequately tailored to be universally described, published, discovered and more importantly composed over the Web. Specifically, service *composition* allows building large-scale evolvable business processes, and stands thereby at the heart of the service paradigm. More specifically, Web-Services are manipulated (e.g. described, published, discovered and composed) using adequate XML-based standards,

including WSDL, UDDI, SOAP [Pap07], WS-BPEL [CGK<sup>+</sup>04] and WS-CDL [KOMC04].

Concerning the adaptability, with these unique features the service paradigm is clearly entrusted with all capabilities to *inherently* promote flexibility and dynamic adaptability, while developing complex cross-organizational applications. Indeed, compared to previous paradigms (e.g. object-[Weg90, OMG05] and component-orientation [OMG01, SG96, MT00]), the service paradigm is endowed with "non-conventional" development techniques steered by the "publish-discover-interact-compose" chain. This flexible advanced development-chain provides all means for resulting in agile and evolving service-oriented applications. First, while publishing services, the provider has the ability to explicitly separate stable service functionalities from other evolving service features such as quality, security or context policies and rules. Furthermore, the discovery process offers the requestors different alternatives to empower it by dynamic selection criteria and different qualities. Third, with the composition as main driving development force, the level of adaptability can be negotiated between the collaborating partners (e.g. providers and requestors). Indeed, while composing (basic) services to built complex business processes and applications, the partners can transparently reason about the qualities of such composition and customize it on-the-fly.

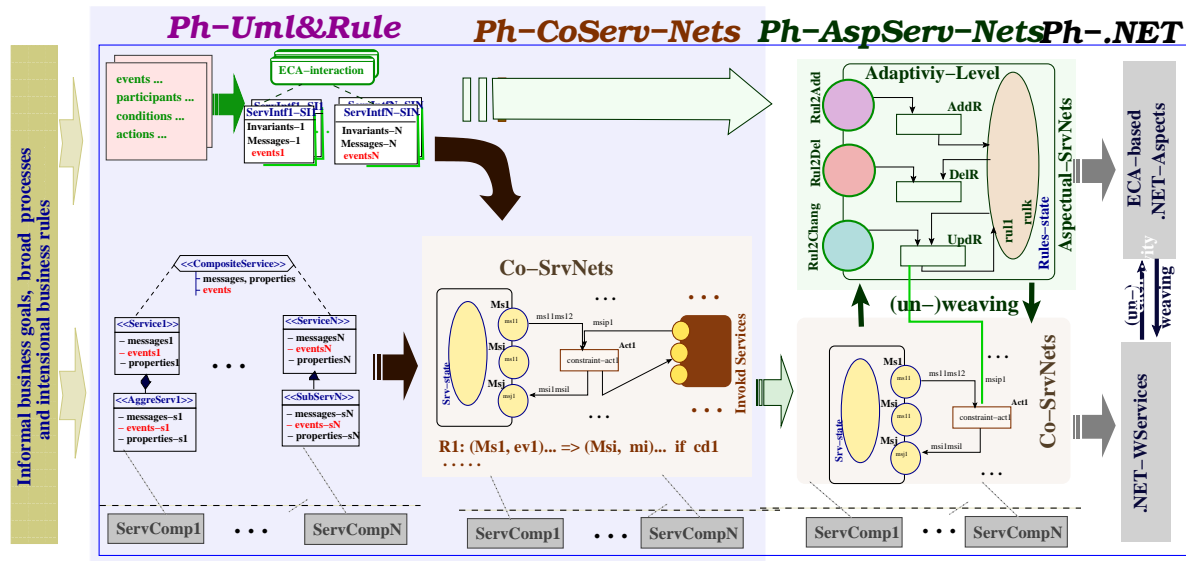


Figure 5.1: Dynamic Adaptability and its Deployment as fourth phase in the forwarded approach

Nevertheless, as we reported in the state-of-art all these potentials, towards resulting in highly adaptive composite realistic services, remain still far from being fully exploited. That is, though both rule-based and aspect-oriented directions are very promising towards promoting adaptability, current proposals remain very akin to the technological and deployment-level. In fact, as we reported in the second chapter, forwarded proposals based on business rules and aspect-orientation focus exclusively on BPEL standard. Furthermore, these approaches support only design-time adaptability. More importantly, both directions do not scale up to the rigorous conceptual-level,



as only means for formally validating and reasoning about the aimed adaptability. Last but not least, the intuitive business-level is not supported by such proposals, which make them very hard for non-expert users.

The purpose of this chapter aims thus at contributing to the dynamic adaptability in a disciplined manner, by pushing forwards the so-far achieved results from the previous chapters. As depicted in Figure 5.1, this chapter focusses on the dynamic adaptability (right-hand side) phase in the forwarded stepwise approach (we discussed in the third chapter). We aim particularly at benefiting from the potentials of aspect-oriented mechanisms in separating conceptual modelling concerns from their dynamic adaptability. More precisely, this chapter addresses runtime adaptability in service-oriented applications, at the conceptual-level by leveraging the CSRV-NETS framework and its rewriting-logic based semantics with rule-driven aspect-oriented concepts. Furthermore, at the service technology-level, we develop a compliant .NET environment that faithfully and efficiently implements this envisioned extension of CSRV-NETS towards run-time adaptability. Before delving into different conceptual peculiarities of leveraging the CSRV-NETS framework with an adaptability-level based on aspect-oriented mechanisms, the next section sheds more light on the achieved (design-time) adaptability in CSRV-NETS framework as well as the potential benefits we expect from the envisioned extension.

## 5.1 CSRV-NETS Design-time service Adaptability: Potentials and Flaws

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-oriented Petri nets formalism and its choreographical extension CCSRV-NETS, allows for intrinsically and soundly integrating these intra- and cross-services governing business rules.

We should further recall that this service formalization is adaptive by construction, since (event-driven ECA) business rules are by nature adaptive and evolving as they reflect business policies and strategies governing any opportunistic cross-organizational alliance of services. In the so-far forwarded approach's steps, we have thus been supporting adaptability through business rules, which we soundly integrate while conceiving CCSRV-NETS. More precisely, in order to adapt or evolve any business rule with respect to an already specified CCSRV-NETS, we *must* redesign the corresponding transition capturing such business rule. This redesign implies that in order to change any transition, we have to explicitly replace some or all its input / output inscriptions as well as its conditions, with associated ECA (events-conditions-actions) elements of the newly emerging business rule. In other words, the so far addressed rule-centric adaptivity is exclusively achieved at *design-time*.

Before delving into the details about how to leverage such design-time rule-centric adaptability towards dynamic and thus *runtime* adaptability, we judge essential to recall some of the benefits of such design-time evolution, through the so-far forwarded CCSRV-NETS framework. Firstly, as we reported from the state-of-art, no existing approach has tackled business rules in service-oriented applications at the *rigorous* conceptual-level (and the semi-formal). Secondly, we have been coping with both intra- and cross-service business rules in a harmonious complementarity. Thirdly, due to the CCSRV-NETS true-concurrent rewriting-logic based semantics, business rules governing different activities / transitions behavior can be simultaneously checked and executed. Fourthly, while changing a given business rule governing a transition behavior, all other activities / transitions may be kept running through their rewriting rules. Furthermore, even for a transition under design-time change, we can always perform it with respect to the current business rule (i.e. before confirming the emerging 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* evolution. 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 service components still 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 ideas of this runtime adaptability and its smooth yet sound conceptualization, we present some of its basic principles, potentials and advantages.

**Explicit separation of adaptability concerns** : Achieving automatic runtime adaptivity in a non-intrusive and unanticipated manner requires, on the one hand, an explicit separation between the running CCSRV-NETS conceptual-level under current business rules and the adaptive-level or aspectual-level<sup>1</sup>, where the rules have to be independently managed. Moreover, at the explicit aspectual-level, all kind of rules (i.e. current, planned and even unanticipated) have to be adequately and dynamically manipulated (e.g. removed, updated or added). On the other hand, runtime adaptability implies the ability to dynamically weave / unweave any business rule on the running CCSRV-NETS conceptual model. Subsequently, we adopt interchangeably adaptive- and aspectual-level terminology, since we are capitalizing on aspect-oriented concepts for the purpose of dynamic adaptability.

**Explicit focus on rules and their evolution** : With the clean separation of business rules at the aspectual-level from the running CCSRV-NETS base-level service conceptualization, (cross-)organization stake-holders can exclusively and independently focus on managing adaptability and thereby enhancing competitiveness. That is, business people become ex-

---

<sup>1</sup>In the literature different terms for such explicit separation, including meta-, reflection-level [CGS02] or more recently aspectual-level [Kea97, EFB01]

clusively responsible for coping with adaptability policies and strategies, whereas software designers focus on how to formalize, implement and dynamically integrate such emerging rules.

**Competitiveness though agility** : As we just emphasized, by empowering the conceptual-level with an extra aspectual-level for handling business rules, we are promoting services to respond to any market changes, such as new emerging competitive policies, and / or new (rule-driven) attractive composition or opportunistic alliance with other services.

**Rules personalization to specific customers and context** : Besides adapting rules due to emerging policies and market changes, dynamic adaptability also directly facilitates the personalization to specific instances of customers / providers and of times and locations. More precisely, since CSRV-NETS promote both type- and *instance*-level modelling, a variety of business rules can be dynamically associated to a given transition. Each business rule can further be tailored to the specificities of requestor / provider instances. The rules can be depending on time such as specific days (week-days and week-end) or months (summer / winter sales and discounts), or specific hours (nights, mornings). Last but not least, customer and provider (moving) locations can play crucial adaptability factor through specific location-dependent rules [AFO06].

**Aspect-oriented MAUDE for runtime adaptability** : For formally validating and reasoning about the dynamic shifting up and down of ECA-driven rules between the aspectual-level and the conceptual-level, we propose to endow MAUDE service components with aspect-oriented mechanisms reflecting the semantics of the envisioned CSRV-NETS aspectual-level. We particularly recapitulate on rewriting logic reflection capabilities, for dynamically intercepting events and dynamically performing and weaving invoked rules from the aspectual-level.

**Compliant .NET Environment** : Towards further enhancing the practicability of the disciplined approach we are proposing for dynamic adaptability, we present its faithful translation at the service technology-level. More precisely, benefiting from the advanced capabilities of the Web-Services .Net environment and its ability to integrate aspect-oriented concepts, we developed a compliant environment that reflects the envisioned conceptual-level.

The remaining sections of this chapter are organized as follows. In the next section, we informally motivate and present the main ideas and principles of the aimed dynamic adaptability in CSRV-NETS. In the third section, we put forward the formal setting for this aspect-oriented leveraging of CSRV-NETS towards dynamic adaptability. In the fourth section we illustrate these conceptualization ideas on the already Flight service CSRV-NETS specification, by smoothly leveraging it towards aspect-oriented rule-centric dynamical adaptability. In the fifth section, we develop on the MAUDE tailoring for governing the proposed aspect-oriented adaptive CSRV-NETS formal-

ism. In the sixth section, we report on the developed compliant .NET environment for efficiently and dynamically adapting Web-Services.

## 5.2 CSRV-NETS-based Aspectual-level: Main Ideas and Concepts

First it is important to point out that using CSRV-NETS structural and behavioral 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 CSRV-NETS model. Moreover, we can update any business rule ingredients (i.e. events / messages / conditions) by updating the arc-inscriptions and the condition of the corresponding transition. However, what goes beyond the hitherto CSRV-NETS conceptualization is the ability of dynamically manipulating any existing business rule governing a given transition. Beyond CSRV-NETS capabilities belong further 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 main ideas and principles to smoothly leverage CSRV-NETS, so that it can address such dynamic adapting of transition behaviors.

### 5.2.1 CSRV-NETS-transitions: Towards an "aspect"-*representation*

Recalling again that any ECA-driven (architectural) rule governing a service-oriented business activity is modelled in CSRV-NETS as transition's behavior. Consequently, the first step towards transparently and dynamically manipulating such ECA-driven rules consists in *externalizing* corresponding CSRV-NETS transitions behavior in a form similar to aspect-oriented *advices*. Towards achieving such externalization and conceptualizing thereby ECA-driven rules as cross-cutting concerns, we should therefore bring a satisfactory answer to the following inherent question:

*"Is it possible to come up with an appropriate "aspect-oriented" representation that permits externalizing any individual CSRV-NETS transition behavior"*

Towards proposing an adequate transition's representation and thereby answering that question, we should first understand what are the ingredients composing any CSRV-NETS transition's behavior. As systematic response, we know that any general CSRV-NETS transition's behavior is definitely composed of: (1) a transition identifier or name; (2) input arc-inscriptions with their corresponding input places; (3) output arc-inscriptions with their corresponding output places; and finally (4) the transition condition. Consequently, a straightforward candidate to explicitly represent any CSRV-NETS transition's behavior may consist in *gathering* these four elements into a single *tuple*. Besides that, since we are aiming at changing such behavior with the possibility of several "versions", we judge beneficial to anticipate enriching such tuples with a fifth element, as a natural counter reflecting the version identity of any particular behavior (for the considered transition).

More precisely, we suggest a straightforward "user-sugar" transition's behavior representation (as aspect-oriented advice) consisting of a five-element tuple of the form:

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

Recalling again that the Input (resp. Output) inscriptions consist of all pairs "input (resp. output) places with associated arc-inscriptions" entering (resp. leaving) that associated transition.

To bring this "abstract" representation one step closer to the specificities of the CSR-V-NETS framework, let us first re-call the generic pattern for CSR-V-NETS transitions. As discussed in the third chapter, we again re-depict in Figure 5.2 that most general form of CSR-V-NETS transitions. They allow bringing into contact different imported messages and events, denoted by  $\otimes_{k=1} Evn_k$  and  $\otimes_{i=1} Ms_i$  to targeted service instance states  $\otimes_{i=1}^k \langle Sid_i | prs_i \rangle$ . Under specific conditions involving message parameters and service state properties, the general effect of such contact corresponds to the consumption of imported messages, the emerging of some new exported messages (denoted by  $\otimes_m Ms'_m$ ) and the change of some properties of involved service states.

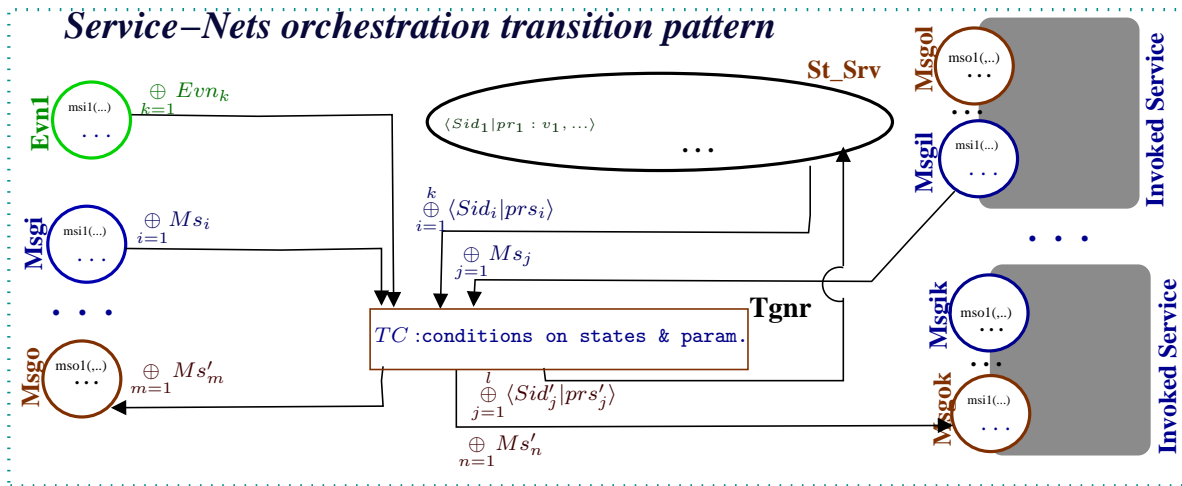


Figure 5.2: The generic CSR-V-NETS-transitions behavior

The projection of the above general transition's representation as independent aspect-oriented advice, composed of the five-element tuple, on this specific CSR-V-NETS-tailored generic transition, results in the following more concrete five-element tuple.

$$\langle Tgnr : v \mid \otimes_e (Env_e, Ev_e) \otimes_{p=1}^n (Msg_p, Ms_{i_p}) \otimes (StSrv, \otimes_{i=1}^k \langle Sid_i | prs_i \rangle), \\ (StSrv, \otimes_{j=1}^l \langle Sid'_j | prs'_j \rangle) \otimes_{q=1}^m (Msg_q, Ms_{o_q}), TC(Tgnr) \rangle$$

where:

- $Tgnr$  represents a transition label or identifier. As we are aiming at updating 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 behavior. Any other adaptation will be referenced by incrementing  $v$  by one (1). We are thus using the version numbers to keep *track* of different transition's changes, so that we can monitor the evolution.
- The third element of the tuple  $\bigotimes_{p=1}^n (Msg_p, Ms_{i_p}) \otimes (StSrv, \bigoplus_{i=1}^k \langle Sid_i | prs_i \rangle)$  defines different input messages and state places with their corresponding (multiset of terms) input-arc inscriptions, as given in the generic transition in Figure 5.2. Important to notice here is that such elements may contain *variables*, exactly like arc-inscriptions associated with usual transitions. That is, they are not like usual (base-level) tokens which must be ground terms (e.g. without any variables).
- The fourth element  $\bigotimes_{j=1}^k (StSrv, \bigoplus_{j=1}^k \langle Sid_j | prs_j \rangle) \bigotimes_{q=1}^m ((Msg_q, Ms_{o_q})$  captures 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 5.2.1** Let us reconsider part of the Flight CSRV-NETS specification from chapter three as depicted in Figure 5.3. Let us further 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 this reduced Flight CSRV-NETS specification, could be represented as follows:

$$\begin{aligned} & \langle Tflg\_bk \quad : \quad 0 \mid (\text{Flight\_Request}, FlgBk(Cs, R, Dy, Py)) \otimes (\text{Flight\_St}, \langle FG | FgInf \quad : \\ & [R.Fr.To.Dt.Tm.Cx], Rsv.Rs, Cmf : Fm \rangle) , \\ & (\text{Flight\_Bookd}, FlgBkd(Cs, R.Fr.To.Dt.Tm, Py)) \otimes (\text{Flight\_Pay}, FlgPay(Cs, R, Py) \wedge \\ & FlgPnl(Cs, R, Pn)) , (Dc \leq Dy) \wedge (Cs \in Rs) \wedge (Py = Cx) \wedge (Pn = 0) \wedge (Cmf.[Cs.R]) \wedge ((Dc \geq \\ & Dy) \wedge (Pn := Py * 0.1)) \rangle \end{aligned}$$

That is, first since this transition behavior is a default one we assign 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.

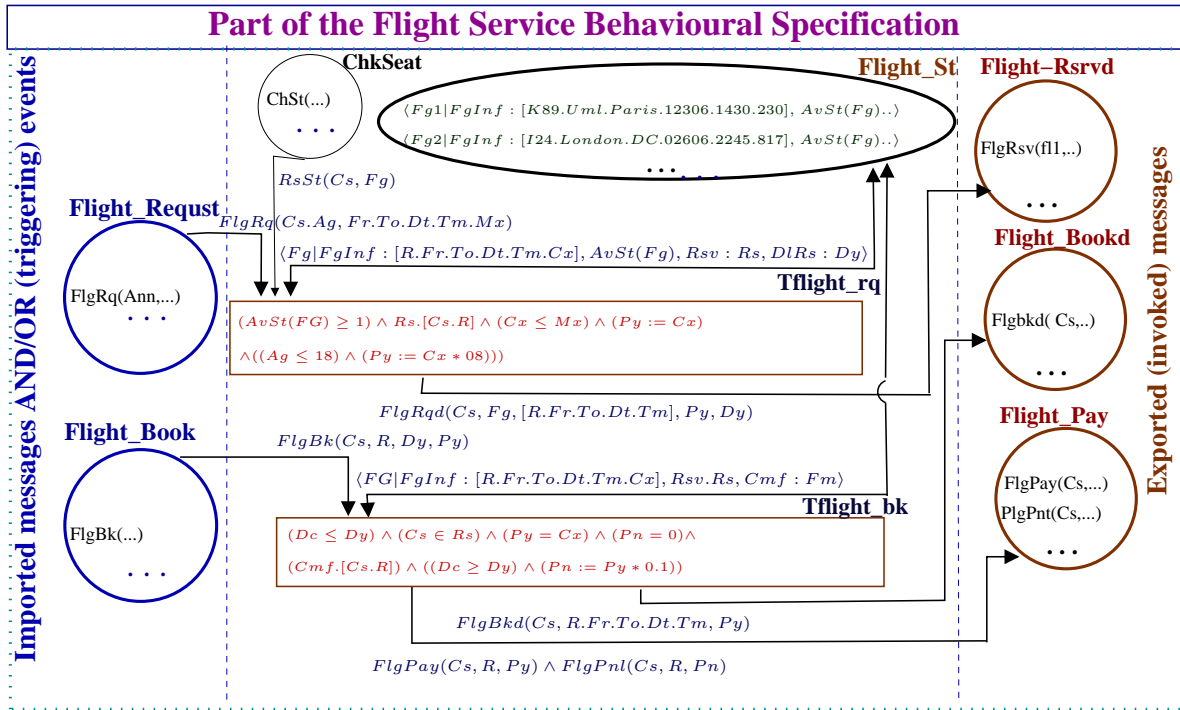


Figure 5.3: Part of the Flight Service CSRV-NETS behavioral Specification.

### 5.2.2 CSRV-NETS-based *aspectual*-Level: Informal presentation

After demonstrating how to capture any CSRV-NETS transition-behavior as advice (i.e. five-element tuple), the next decisive step concerns the independent and dynamic management (e.g. updating, adding and removing) of such advices. Nevertheless, to stay compatible with the CSRV-NETS framework, such manipulation should be geared by (high-level) Petri nets; otherwise it would be impossible to dynamically connect the CSRV-NETS conceptual-level with such envisioned aspectual-level. More precisely, the Petri nets-based proposal we are looking for to realize such management, could be summarized in the following steps:

**Aspectual-place for gathering the advices-as-rules :** At first we propose to *gather* such ECA-driven rules and transition-behaviors into an associated "*aspectual*" state-rule place. Such aspectual rule-place reflects thus the first construct of the envisioned Petri nets-driven aspect-oriented adaptability, namely ECA-driven rules-as-tokens. That is, this aspectual rule-place allows keeping all existing and emerging ECA-driven driven rules (i.e. advices-as-tuples).

Given such advices within that aspectual rule-place, we require then further aspectual-places and -transitions for effectively manipulating them.

**Places for triggering changes :** Since we are looking for updating / modifying, adding

and/or removing any tuple, we propose three corresponding (aspectual-operation) places, we denote by  $R12Del$ ,  $R12Add$  and  $R12Chg$ . These (aspectual-)places permit thus for storing any corresponding (aspectual-)messages for changing, adding or removing any tuple. As will be detailed later, we may denote such aspectual-messages with respectively  $Chg\_R1$  (Rule-as-tuple),  $Add\_R1$  (Rule-as-tuple) and  $Del\_R1$  (Rule-as-tuple).

”**Aspectual**” **Transitions for manipulating the rules-as-tuple** : Given the above ”aspect-oriented” places with their tokens, the last step for effectively enabling any dynamic manipulation of ECA-driven rules consists in conceiving three associated aspectual-transitions. We denote such aspectual transitions as  $RL2DL$ ,  $RL2AD$  and  $RL2CHG$ . They relate the places  $Chg\_R1$ ,  $Ad\_R1$  and  $DL\_R1$  to the aspect-oriented state-rule place denoted  $RsP$ . Such transitions permit thus selecting any specific ECA-driven rule-as-tuple and allow dynamically manipulating it (e.g. change / add / deletion) as-tuple.

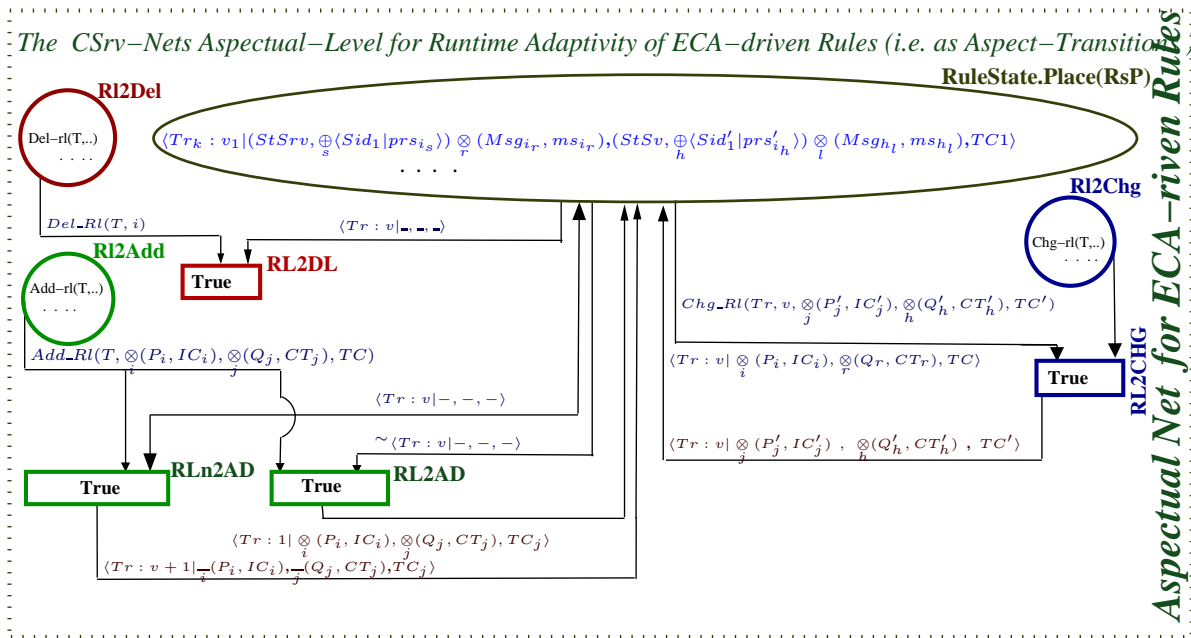


Figure 5.4: The CSrv-NETS-based Aspectual Net for ECA-driven Rules dynamism

This CSrv-NETS-aspectual-level is graphically made more clear as clarified in Figure 5.4. That is, the three main proposed aspect-oriented principles are reflected, as milestones towards dynamic adaptability of ECA-driven rules captured as transition’s behavior using tuples. More precisely, this proposed general aspectual adaptability-level is again a specific variant of high-level Petri nets, but with tokens including variables, as they capture non-instantiated ECA-driven rules. We further note that for adding any ECA-driven rule as (transition-)behaviors, we have to distinguish two complementary cases. The first case concerns the situation where the to-be transition-behavior corresponds to a completely new ECA-driven rule; that is such transition does not already exist



in the aspectual state-rule place. For that purpose, we resort to the symbol  $\sim$  to check this non-existence i.e. as inhibitor-arc. In this case, the version number is set to the initial number one (1). The second case concerns the adding of the new version behavior to an existing transition, where we have to increment the version counter by one.

### 5.2.3 CSRV-NETS-based *aspectual*-Level: Formal setting

After this informal presentation, we bring in the following this CSRV-NETS-based aspectual-level into a more rigorous stand. For that and in the same spirit as CSRV-NETS, we first require some preliminaries notations and concepts. More precisely, we first introduce and define the notion of (CSRV-NETS-based) aspectual-template signature, that formalizes the algebraic structures for the rules-as-tuple and the aspectual operations for their dynamic manipulation. We then fix the notations for different algebraic terms, governed by such aspectual template signature. Last but not least, the notion of CSRV-NETS aspectual-level is formally defined.

**Definition 5.2.2 (Aspectual-template signature)** We assume given a CSRV-NETS specification as given in definition 3.4.2, namely a structure  $(SvP, SvT, SvPre, SvPost, s, SvTC)$  modeling a service component  $sc_i$ . The aspectual-template signature is then defined as a pair  $\{ASt_{rl}, AD_{rl}, MD_{rl}, AD_{rl}\}, \{ASt_{op}, Ad_{rl}, Md_{rl}, DL_{rl}\}$  with:

- $ASt_{rl}$  (i.e. Aspectual-State) represents the sort for capturing the rules-as-tuple. The sorts  $AD_{rl}, MD_{rl}, AD_{rl}$  are similarly introduced for capturing the aspectual-level operations for adding, modifying and deleting such rules-as-tuples respectively;
- $ASt_{op}$  is the aspectual-state constructor operation, reflecting the five-element tuple. That is, this operation is indexed by  $T \times \mathbb{N} \times SvPre \times SvPost \times SvTC \times ASt_{op}$ . As we discussed above, we denote this operation as equivalent five-element tuple indexed by the mix-fix notation:  $\langle \_ : \_ | \_ , \_ , \_ \rangle : SvT \times \mathbb{N} \times SvPre \times SvPost \times SvTC \rightarrow ASt_{op}$
- The (message) operation  $Ad_{rl}$  for adding ECA-driven rules as-tuple, is indexed by  $SvT \times SvPre \times SvPost \times SvTC \times AD_{rl}$ . To update such behavior as-tuple, we adopt the message operation  $Md_{rl}$ , is indexed by  $SvT \times \mathbb{N} \times SvPre \times SvPost \times SvTC \times MD_{rl}$ . Finally, to remove any ECA-driven rule as-tuple, we adopt the message operation  $DL_{rl}$ . It is indexed by  $SvT \times \mathbb{N} \times DL_{rl}$ .

Please note that for sake of simplicity, while defining different operations we are adopting  $SvPre$ ,  $SvPost$  and  $SvTC$ , instead of their effective target sorts. That is in the above definition  $SvPre$  and  $SvPost$  should be understood as the multiset sort  $[MT(X)]_{\otimes}$  induced by  $[SvP \times [T_{Sv}(X)]_{\oplus} \cup [T_{Ms}(X)]_{\oplus}]_{\otimes}$ . Whereas  $SvTC$  should stand for the boolean sort, ranging over  $(T_{Sv}(X) \cup T_{Ms}(X))_{bool}$ . Given such CSRV-NETS-based aspectual-template signature, we can define over it any derived algebraic terms as follows.

**Definition 5.2.3 (Terms generated by the Aspectual-template signature)** Given a (CSRVS-NETS-based) aspectual-template signature as defined above, we then introduce and define the following generated algebraic terms.

- (1) We define and denote by  $\mathcal{T}_{Ast_{rl}}(\mathcal{X})$ , the set of terms of sorts in  $Ast_{rl}, AD_{rl}, MD_{rl}$  and  $AD_{rl}$  over a set of (aspectual-)variables, which is denoted by  $\mathcal{X}$ . In this sense, a term of this sort allows capturing any specific ECA-driven as-tuple, any addition, modification or deletion specific message for such tuples. We further assume that this variables set can be split into the following sub-sets:  $\mathcal{X}_{IC}^i, \mathcal{X}_{CT}^i$  and  $\mathcal{X}_{TC}^i$  with respective sorts  $Pre, Post$  and  $TC$ . To precisely capture some details, instead of using these global variables, we will often use derived variables, we denote by  $SvP_i^v, IC_i^v, CT_i^v, CT_i^v$  to respectively refer to the sorts  $SvP$  (i.e. base-level CSRVS-NETS-places),  $[T_{SvO}(X)]_{\oplus} \cup [T_{Msg}(X)]_{\oplus}$  and the boolean. In this sense, for instance,  $\mathcal{X}_{IC}^i$  will be an abbreviation to be instantiated by  $\otimes_i(SvP_i^v, IC_i^v)$ .
- (2) We will denote by  $\mathcal{BT}(\mathcal{X})$  the multiset over  $\mathcal{T}_{Ast_{rl}}(\mathcal{X})$  induced by the internal associative commutative multiset operation we denote by  $\ominus$ . Equivalence classes induced by this multiset will be denoted by  $[\mathcal{T}(\mathcal{X})]_{\ominus}$ .
- (3) To capture different arc inscriptions and meta-net states, another multiset denoted  $\mathcal{MT}(\mathcal{X})$  over  $SvP_{rl} \times [\mathcal{T}_{Ast_{rl}}(\mathcal{X})]_{\ominus}$  is introduced. We denote by  $\otimes$  the union operation over this multiset, and its elements will be abstracted by  $[\mathcal{MT}_{Ast_{rl}}(\mathcal{X})]_{\otimes}$ .  $SvP_{rl}$  represents places of the meta-net precisely defined in what follows.

With these preliminary notations and definitions, in the following we formalize the concept of marked **aspectual-net** related to given CSRVS-NETS specification.

**Definition 5.2.4 (CSRVS-NETS-based Aspectual-Net)** We assume given an aspectual-template signature as above defined. An (CSRVS-NETS-based) Aspectual-Net is a marked Petri net structure composed of  $(AsP_{rl}, Ast_{rl}, AsPre_{rl}, AsPost_{rl}, as_{rl}, AsTC_{rl}, AfM_{rl})$  with:

- $AsP_{rl}$  a set of places composed of four (aspectual-)places. A possible notation for these places is:  $AsP_{rl} = \{AsP_{sc_i}, ad_{rl_{sc_i}}, md_{rl_{sc_i}}, dl_{rl_{sc_i}}\}$ .
- $as_{rl} : AsP_{rl} \longrightarrow \{Ast_{rl}, AD_{rl}, MD_{rl}, AD_{rl}\}$  is a bijection associating with each place its related sort. More precisely  $as_{rl}(AsP_{sc_i}) = Ast_{rl}$ ,  $as_{rl}(ad_{rl_{sc_i}}) = AD_{rl}$ ,  $as_{rl}(md_{rl_{sc_i}}) = MD_{rl}$  and  $as_{rl}(dl_{rl_{sc_i}}) = DL_{rl}$ .
- $AsT_{rl} = \{AsT_{Ad_1}, AsT_{Ad_2}, AsT_{Md}, AsT_{Dl}\}$  is a set of transitions for respectively adding, modifying or deleting any ECA-driven rule as-tuple.
- $AsPre_{rl} : AsT_{rl} \longrightarrow [\mathcal{MT}(\mathcal{X})]_{\otimes}$ , with:
  - $AsPre_{rl}(T_{Ad_1}) \in (ad_{rl_{sc_i}}, [\mathcal{T}_{Ad_h}(\mathcal{X})]_{\ominus}) \otimes (AsP_{sc_i}, [\mathcal{T}_{M_h}(\mathcal{X})]_{\ominus})$ ;
  - $Pre_{rl}(T_{Ad_2}) \in (ad_{rl_{sc_i}}, [\mathcal{T}_{Ad_h}(\mathcal{X})]_{\ominus}) \otimes (AsP_{c_i}, [\mathcal{T}_{M_h}(\mathcal{X})^{\sim}]_{\ominus})$ ;
  - $Pre_{rl}(T_{Md}) \in (md_{rl_{sc_i}}, [\mathcal{T}_{Md_{rl}}(\mathcal{X})]_{\ominus}) \otimes (AsP_{sc_i}, [\mathcal{T}_{AsM_{rl}}(\mathcal{X})]_{\ominus})$ ;

- $AsPre_{rl}(T_{Dl}) \in (dl_{rl}_{sci}, [T_{Dl_{rl}}(\mathcal{X})]_{\ominus}) \otimes (AsP_{sci}, [T_{AsM_{rl}}(\mathcal{X})]_{\ominus})$ ;
- $AsPost_{rl} : AsT_{rl} \rightarrow [MT(\mathcal{X})]_{\ominus}$ , with:  $\forall t \in AsT_{rl}, AsPost_{rl}(t) \in (AsP_{sci}, [T_{M_{rl}}(\mathcal{X})]_{\ominus})$
- $AsTC_{rl} : T_{rl} \rightarrow \mathcal{T}(\mathcal{X})_{bool}$  is a boolean expression associated with each transition in  $T_m$ .
- $AfM_{rl} : AsP_{rl} \rightarrow [T_{Ast_{rl}}(\emptyset)]_{\ominus}$  is a marking function such that,  $\forall p \in AsP_{rl}$  we have  $AfM_{rl}(p) \in [AfT_{s(p)}(\emptyset)]_{\ominus}$ .

### 5.3 CSRV-NETS meets its Aspectual Net: Jointpoints and pointcuts at concerns

So far we motivated and presented how to conceive any (CSRV-NETS-based) aspectual-level net, for dynamically adapting any ECA-driven rule, with respect to any supposedly existing CSRV-NETS. The next crucial step concerns thus the concrete connection of this aspectual-level net to a really existing and running base-level CSRV-NETS model. Indeed, only after a satisfactory conceptualization of such linking between the CSRV-NETS base-level and its corresponding aspectual-level, we can speak about the dynamic weaving / unweaving (i.e. shifting-down / shifting-up) of any emerging ECA-driven rule on running CSRV-NETS services.

Recalling that in terms of aspect-oriented concepts [Kea01, EFB01], the synergic linking between a running base-level and its aspectual-level corresponds to the precise conceptualization and mechanization of different pointcuts and jointpoints. That is, whereas the jointpoints define precisely *where* to inject or weave any aspectual advice, the pointcuts permit composing and reasoning about different advices and how to facilitate their (un)weaving at different jointpoints. We should also remind that our advices are different ECA-driven rules represented as five-element tuples, supposedly referring to specific CSRV-NETS base-level transition's behaviors.

#### 5.3.1 CSRV-NETS and its smooth Endowing with Jointpoints

Towards preparing any CSRV-NETS service component to become adaptive, that is, able to dynamically receive any aspectual advices, we have to endow it with "extra" conceptual primitives as jointpoints. Nevertheless, towards coming up with judicious both fine- and coarse-grained yet non-intrusive jointpoints, several requirements should be fulfilled while conceiving them. First, the proposed jointpoints should be so seamless and light that the original CSRV-NETS service component remain practically unchanged and keeping running. In other words, the enriched CSRV-NETS with jointpoints should look and behave as if nothing has been added up. Second, the proposed jointpoints should facilitate both a fine- as well as coarse-grained. More precisely, since the aspectual advices consist of ECA-driven rules, the jointpoints should besides adapting any rule as a unit also allow separately adapting any composing ECA-rule elements, such as the triggering events and /or the related constraints and / or the applied actions. Third, the jointpoints should promote any adopted pointcut-based composite strategy, such as advice disjunctions, conjunctions, choice,

sequence, concurrency, etc.

In the following, we informally bring through the main ideas and the mechanisms underlying such envisioned CSRV-NETS-based joinpoints. Recall again that any ECA-driven rule is reflected at the CSRV-NETS base-level as a transition's behavior with all its ingredients. In other words, our joinpoints should be conceived exclusively around CSRV-NETS transitions, that is, CSRV-NETS places remain completely unchanged. More precisely, the proposed joinpoints are expressed in terms of the following conceptual constructions, that should enrich any selected base-level CSRV-NETS transition towards endowing it with adaptability.

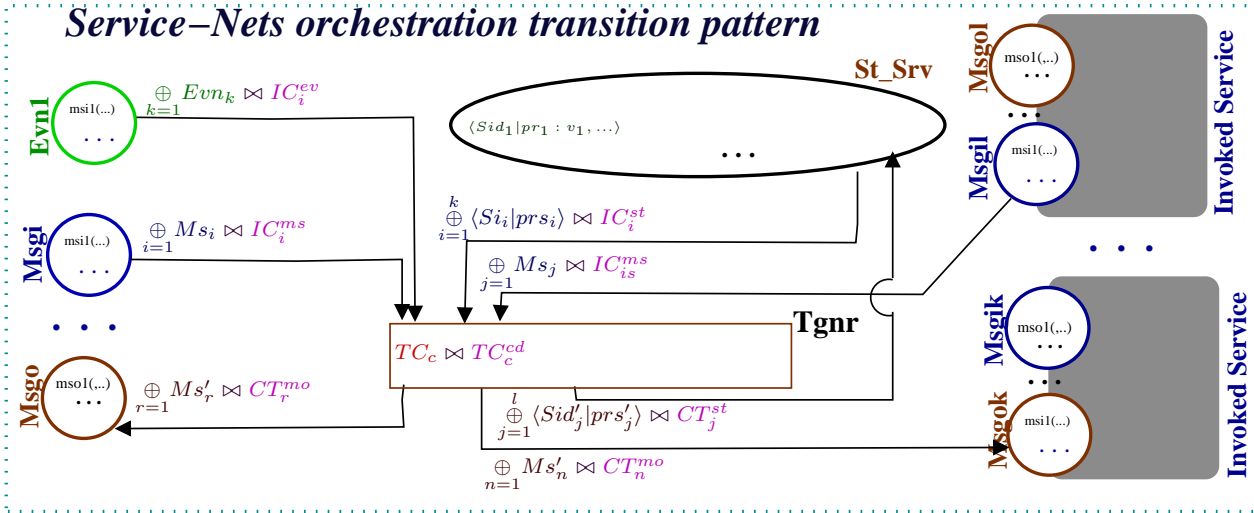


Figure 5.5: Leveraging (generic) CSRV-NETS-transitions with aspectual-variables as Joinpoints

**Extra (aspectual-)variables for Joinpoints** : To prepare runtime (un)weaving of any ECA-driven rule as transition's advice, we propose to smoothly leveraging any concerned and thus to-be adaptive CSRV-NETS-transition as follows. We introduce specific (aspectual-)variables ranging their sorts over different input / output arc-inscriptions and / or condition of such transition. As we already discussed in definition 5.2.3, we denote such aspectual-level variables as  $IC_i^v$ ,  $CT_i^v$ ,  $TC_i^v$ —to respectively referring input and output created tokens and condition parts.

**Joinpoints as "Aspectual" CSRV-NETS-transitions** : The next step in defining joinpoints consists in consistently *composing* such conceived aspectual-variables with existing transition's input / output arc-inscriptions and conditions. We denote the corresponding composition operator by  $\bowtie$ . Intuitively speaking as we detail later, this new composition operator will be semantically playing different meanings such conjunction ( $\wedge$ ) and / or disjunction ( $\vee$ ) of the to-be woven behavior with existing running one at that transition. Subsequently, we refer to such slightly enriched CSRV-NETS-transitions simply as aspectual (CSRV-NETS) transitions.

**Joinpoints applied on generic CSRV-NETS-transitions** : As illustrated through Figure 5.5, the application of these conceptual constructions results in a slightly leveraging the generic pattern of CSRV-NETS-transitions towards such aspectual transitions. That is, different triggering events and input messages (i.e.  $Ev_{i_1}$  and  $ms_{i_1}$ ) are to-be enriched with respective coherent aspectual-variables (i.e.  $IC_{i_1}^{ev}$  and  $IC_{i_1}^{ms}$ ). They are thus enriched through the composition operator  $\bowtie$ , leading to "updatable" input arc-inscriptions, namely  $Ev_{i_1} \bowtie IC_{i_1}^{ev}$  and  $Ms_{j_1} \bowtie IC_{j_1}^{ms}$ . The same reasoning applies to the other transition inscriptions, such as the output and condition parts, as detailed in the Figure.

► **Remark 5.3.1** It is important to emphasize that this smooth shifting from a usual rigid CSRV-NETS-transition towards a corresponding to-be-adaptable aspectual transition is highly flexible on several aspects. First, the shifting could be applied to any transition and at any running time, both forth and back; that is, an already leveraged to an aspectual transition could become rigid (resp. partially adaptable) by removing all (resp. specific) added aspectual variables. More particularly, depending on the specificities of the application-at-hand and current circumstances of the specified CSRV-NETS specification and the surrounding environment, the designer can leverage any rigid transition to an aspectual one and vice-versa. As we mentioned, we can further decide for a "partial" aspectual transition, that is, we decide enriching only specific input and / or output and / or condition judged to be volatile and adaptive.

### 5.3.2 Pointcuts for Connecting CSRV-NETS-Joinpoints to the Aspectual Net

By adopting a Petri nets-based aspect-oriented conceptualization, the connection between the enriched CSRV-NETS-transitions and the aspectual net should be graphically supported. In such manner, the weaving of advices from the aspectual net to the CSRV-NETS base-level become explicit and graphically animated. We should here point out that in usual (textual) aspect-oriented programming languages, the weaving of advices is implicit and internally implemented in the respective compiler. This hidden (aspectual- and base-levels) connection presents several limitations including: (1) The inability to adapt and reason about it in transparent manner; (2) The difficulty of understanding the weaving process by non-experts to such languages; and the impossibility to graphically animate and validate such weaving explicitly.

More precisely, as depicted in Figure 5.6, we are proposing *read-arcs* as basic pointcuts<sup>2</sup>. Such read-arcs allow thus relating any enriched aspectual CSRV-NETS-transition (with joinpoints) to the aspectual-level rule-place, where tokens represent emerging ECA-driven rules as tuples. This explicit graphical syntactical connection facilitates the shifting up and down of (different elements of) ECA-driven rules from the aspectual-level net to any running base-level enriched CSRV-NETS transition.

---

<sup>2</sup>More complex composite pointcuts can be formulated while (un-)weaving advices, as we address next.

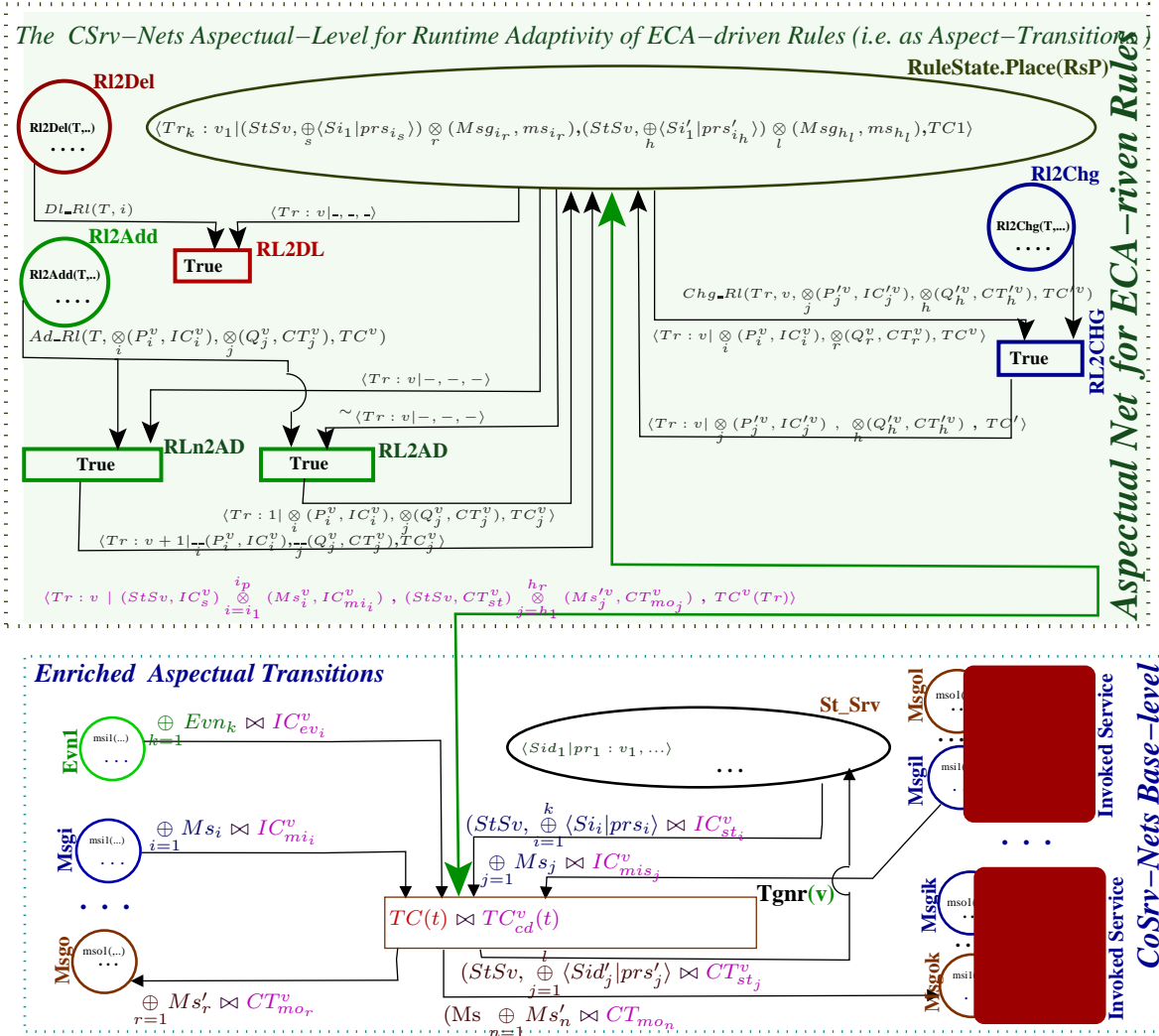


Figure 5.6: Generic transitions for the two-level Aspectual AOCSRV-NETS formalism

What concerns the inscriptions to attach to these read-arcs relating transitions to the aspectual rule-place, we should respect two important requirements. First, the arc-inscriptions must be compliant with the five-element ECA-driven rules as-advice. That is, the terms attached to such read-arc inscriptions must belong to  $[\mathcal{T}_{AStr_l}(\mathcal{X})]_{\ominus}$  (as given in definition 5.2.3). Second, towards a coherent and meaningful propagation of any advice, we have to use in these read-arc inscriptions the *same* aspectual-variables adopted for leveraging the base-level transitions. More precisely, as depicted at the middle of Figure 5.5, the resulting read-arc inscription, relating the aspectual-level state rule-place to the enriched generic CSRV-NETS transition, takes the following form:

$$\langle Tgnr : v \mid (P_s^v, IC_s^v) \underset{i=i_1}{\overset{i_p}{\otimes}} (Ms_i^v, IC_{mo_i}^v), (Q_s^v, CT_s^v) \underset{j=h_1}{\overset{h_r}{\otimes}} (Ms_j^v, CT_{mo_j}^v), TC^v(Tgnr(v)) \rangle$$

That is, besides the transition label **Tgnr** and the variable for keeping track of the version  $v$ , all input (resp. output) places are paired with their respective *aspectual-variables* from corresponding enriched base-level arc-inscriptions. Please note that  $P_s^v$  and  $Q_s^v$  refer to service-state place variables. They can be replaced by any service-state place, which stands in the orchestration-case to just one place we have been denoted by  $StSrv$  in Figure 5.5. Nevertheless, we should point out in the general choreographical composition case. That is, when several service states come into play, the above pair place-inscription, i.e.  $(P_s^v, IC_s^v)$ , should be generalized to  $\underset{s}{\otimes}(P_s^v, IC_s^v)$ . The same applies to the associated output inscription. On the other side, input (resp. output message place) variables are denoted by  $Ms_-^v$  (resp.  $Ms'^v_-$ ). They can thus be substituted by any message or event place at the base-level.

### 5.3.3 AOCSR-NETS: Aspect-oriented CSRV-NETS-extension Formalization

Recalling that in aspect-oriented programming, any AOP language is defined as an indivisible semantical entity, and not as two explicit separated levels as we have been so-far doing. Capitalizing on the previous conceptualization, we therefore present in this subsection a *unified* integrated formal definition of this aspect-oriented extension of CSRV-NETS, that we refer to as AOCSR-NETS. That is, instead of the explicit separated definitions CSRV-NETS and its aspectual net, we come up with a single formalism that serves both the base- and aspectual-levels as well as the shifting up-down of ECA-rules as advice between them. Recalling before that, we can at any time decide which CSRV-NETS-transitions are to be adaptable and thereby dynamically receiving rules (abbreviated as **r1**) and which should remain rigid or fixed (abbreviated as **fx**). This corresponds simply to dynamically add (resp. remove) the extra-joinpoints to any to-be adaptable (resp. rigid) transition. To formally capture this transition's flexibility, we assume that the set of base-level CSRV-NETS-transitions is composed of two subsets, we denote respectively by  $SvT_{rl}$  and  $SvT_{fx}$ . We further note that the subset for to-be adaptive transitions, should be parameterized to keep track of the versions, that is,  $SvT_{rl}$  should become instead as  $SvT_{rl}(-)$ .

**Definition 5.3.2** (AOCSRv-NETS) We assume given a CSRv-NETS specification as defined in Definition 3.4.2, namely a structure  $(SvP, SvT, SvPre, SvPost, s, SvTC)$  modeling a service component  $sc_i$ . We further assume as given an associated aspectual-net  $(AsP_{rl}, AsT_{rl}, AsPre_{rl}, AsPost_{rl}, as_{rl}, AsTC_m)$  as formalized in definition 5.2.4. We then define an aspect-oriented AOCSRv-NETS from this specification as a structure,  $(P_{AS}, T_{AS}, Pre_{AS}, Post_{AS}, s_{AS}, TC_{AS})$  with:

- $P_{AS} = SvP \cup AsP_{rl}$ .
- $s_{AS}(p) = s(p)$  for  $p \in SvP$  and  $s_{AP}(p) = as_{rl}(p)$  for  $p \in AsP_{rl}$ .
- $T_{AS} = SvT_{fx} \cup SvT_{rl}(\mathbb{N}) \cup AsT_{rl}$ , with  $SvT = \{SvT_{fx}, SvT_{rl}\}$ . To capture the notion of version, each identifier transition in  $SvT_{rl}$  is now parametrized by natural, that is,  $SvT_{rl}(\mathbb{N})$ .
- $Pre_{AS} = \{SvPre_{fx}, SvPre_{rl}, AsPre_{rl}\}$  and  $Post_{AS} = \{SvPost_{fx}, SvPost_{rl}, AsPost_{rl}\}$  with **for**  $t \in SvT_{fx}$  :  $Pre_{AS}(t) = SvPre(t)$  and  $Post_{AS}(t) = SvPost(t)$ . That is, arc-inscriptions for selected-as-rigid CSRv-NETS transitions remain unchanged.
- for**  $t \in AsT_{rl}$  :  $Pre_{AS}(t) = AsPre_{rl}(t)$  and  $Post_{AS}(t) = AsPost_{rl}(t)$ . That is, arc-inscriptions of the aspectual net as preserved as already defined.
- for**  $t(-) \in SvT_{rl}$  :
  - $SvPre_{rl}(t(-)) = (\mathcal{R}f\mathcal{P}, \langle t : - | \otimes_i (p_i^v, IC_i^v), \otimes_j (p_j^v, CT_j^v), TC^v \rangle) \parallel_r \otimes_i (p_i, mt_i \bowtie IC_i^v)$ .  
That is, corresponding original CSRv-NETS transitions are now enriched with read-arc inscriptions and aspectual variables.
  - $Post_{rl}(t(-)) = \otimes_j (q_j, mt_j \bowtie CT_j^v)$ . Likewise output-inscriptions for aspectual to-be adaptable CSRv-NETS transitions are enriched with aspectual variables.
- $TC_{AS} = \{SvTC_{rl}, SvTC_{fx}, AsTC\}$ . That is  $TC_{AS}(t) = SvTC(t)$  for all  $t \in SvT_{fx}$ ,  $TC_{AS}(t) = AsTC_{rl}(t)$  for all  $t \in AsT_{rl}$ , and  $SvTC_{rl}(t(-)) = SvTC(t) \bowtie TC^v$  for all  $t \in SvT_{rl}$

## 5.4 Runtime (un)weaving of advices in AOCSRv-NETS: Principles and Formalization

The last phase in this CSRv-NETS-based aspect-oriented conceptualization towards (ECA-driven) runtime adaptability, consists in defining how aspectual transitions will be dynamically instantiated and executed. In other words, we should detail how to *dynamically* weave and / or unweave any (elements of) ECA-driven rules from the aspectual-level state-rule places  $\mathbf{RSP}$ , using the linked base-level aspectual transitions.

In the following we thus first informally explain this process, and afterwards define it in a more disciplined manner. Let us detailing the meaning of the crucial operator  $\bowtie$ , composing existing CSRv-NETS-transition inscriptions with aspectual-variables as joinpoints. More specifically, with the aim to cover a maximum of compositional cases, we propose to assign to this  $\bowtie$  operator, al-least four semantical interpretations:



- $\bowtie$  *asp-var with asp-var as nil* : Towards promoting flexibility, even after enriching a transition with a joinpoints as aspectual-variables (i.e. **asp-var**), we keep working with the default initial behavior. That is, we may decide to propagate no emerging behavior from the aspectual-level. Indeed, in real-word service-oriented applications, there are always some business activities (modelled here as transitions) required to be fixed. For that reasons, we are allowing in the proposed interpretation to interpret, in such case, all extra inscriptions " $\bowtie$  **asp-var**" as if they do not exist, i.e. as *nil*.
- $\bowtie$  *asp-var as an or ( $\vee$ ) operator* : Interpreting the operator  $\bowtie$  as a choice, that is as  $\vee$  operator, means that for firing the concerned transition, we should dynamically bring down a new behavior from the aspectual state-place. In other words, the old existing initial behavior is to be skipped and dynamically replaced by any new ECA-driven behavior. In this case, the default transition's behavior is *completely* swapped with the new propagated one. Nevertheless, we argue that its presence at the arc-inscriptions permit to remind the designer of that initial behavior as reference for changes.
- $\bowtie$  *asp-var as a conjunction ( $\wedge$ ) operator* : The next possibility we propose to offer while dynamically bringing down any behavior as-tuple, consists in dynamically softening / tightening / enriching any already existing behavior with more knowledge. That is, we propose to interpret the operator  $\bowtie$  as a conjunction ( $\wedge$ ). This implies integrating the newly woven behavior with the existing one. For instance, we may add a new constraint to the condition, and thereby tightening it further. We may also add new input / output messages while involving an existing transition and bring thereby more flexibility.
- $\bowtie$  *as mixed composite operator* : Although we will not detail it further, from the above interpretations it is quite straightforward to consider *within a same transition* all the three discussed possibilities. For instance, we may skip adapting some input messages, while dynamically tightening others via  $\bowtie$  as a conjunction ( $\wedge$ ) and softening some other output messages by interpreting the operator as a disjunction ( $\vee$ ). By convention, we propose to adopt the minus symbol "-" at the corresponding position(s) in the read-arc, when the associated arc-inscription element(s) should remain unchanged. For instance, the following aspectual read-arc  $\langle Tg1 : 1 \mid -, -, TC^t \rangle$  implies that all input and output inscriptions must remain unchanged. Thus only the condition is subject to dynamic changes from the aspectual-level.

Recalling that the aspect-oriented AOCSRVS-NETS is still behaviorally governed by two-level rewrite theory. That is, we are still able to concurrently running any base-level CSRVS-NETS "default" specification using its associated rewrite theory, we developed in the previous chapters. That is, when ignoring read-arcs relating base-level to its aspectual one as well as added aspectual-variables, the governing rules remain unchanged as previously defined. Similarly, at the aspectual-level, we can concurrently adding, modifying or deleting any ECA-driven transition's behaviour as-tuple, using slightly adapted rewrite theory to the specificities of that aspectual-level.

Nevertheless, by taking into account read-arcs and associated aspectual-variables attached to aspect-oriented transitions, neither the base- nor the aspectual-level rewrite theory could be directly and effectively applied. Indeed, these aspect-oriented transitions though defined at the base-level they are semantically bounded to the aspectual-level. To become operational and get fired, aspect-oriented transitions deserve a specific inherent (rewriting-logic based) semantics that judiciously brings into play both base- and aspectual-level rewrite theories.

Towards coming up with such tailored two-level rewrite theory, and its respective inference rules, we proceed progressively as follows. First, we propose to govern the behavior of any aspectual-oriented transition as a rewrite rule using its general pattern as explained and depicted in Figure 5.6. As we just emphasized such transitions rules cannot be directly applied neither at the base- nor at the aspectual-level. For that reason, we will refer to as "non-woven" (aspect-oriented) transition rules. We then come up with mechanisms and tailored inference rules, so such non-woven transition rules become executable, while capturing the dynamic (un)weaving of (elements of) ECA-driven rules from the aspectual-level.

#### 5.4.1 "Non-woven" Rewriting rules governing aspect-oriented transitions

As we just emphasized, as first step towards dynamically weaving and running (elements of) ECA-driven rules via respective aspect-oriented transitions, we propose to directly reflect the behavior of such transitions as rewrite rules, using their input-, output-inscriptions and associated condition. More precisely, we do so with respect to the generic transition depicted in Figure 5.6, as its formal inscriptions (i.e.  $AsPre(-)$ ,  $AsPost(-)$ ,  $TC(-)$ ) are given in AOCSRVS-NETS definition 5.3.2.

More precisely, with respect to the formal definition in definition 5.3.2 of non-instantiated transition, their direct translation into a rewrite rule takes the following form.

**Definition 5.4.1 (Non-woven rewriting rules for Aspectual Transitions)** Given an aspect-oriented AOCSRVS-NETS as defined in definition 5.3.2, the corresponding "non-woven" (shortly **nwv**)rewrite rules of aspect-oriented transitions  $SvT(-)$  can be directly expressed as follows:

$$\begin{aligned} t^{nwv}(-) : (\mathcal{P}, \langle t : - \mid \otimes_i (P_i^v, IC_i^v), \otimes_j (Q_j^v, CT_j^v), TC^v \rangle) \parallel_r \otimes_i (p_i, mt_i \vee IC_i^v) \\ \Rightarrow \otimes_j (p_j, mt_j \vee CT_j^v) \text{ if } TC(Tgnr) \vee TC^v. \end{aligned}$$

With respect to the generic aspect-oriented transition  $Tgnr(-)$  depicted in Figure 5.6, we result in the corresponding specific non-woven rewrite rule:

$$\begin{aligned} Tgnr^{nwv} : (\mathcal{R}f\mathcal{P}, \langle Tgnr : - \mid \otimes_i (P_i^v, IC_i^v), \otimes_j (Q_j^v, CT_j^v), TC^v \rangle) \parallel_r (\text{StSv}, \oplus_s \langle Si_i | prs_s \rangle \bowtie IC_{st_s}^v) \otimes_{i=1} \\ (Msgi_i^v, Ms_i^v \bowtie IC_{mi_i}^v) \\ \Rightarrow (\text{StSv}, \oplus_{s'} \langle Si'_i | prs'_{s'} \rangle \bowtie CT_{st_{s'}}^v) \otimes_{o=1} (\text{Msgo}_o^v, Ms'_o \bowtie CT_{mo_o}^v) \text{ if } TC(Tgnr) \bowtie TC^v(Tgnr). \end{aligned}$$

Recalling that  $Msgi^v$  and  $Msgo^v$  stand for message identifier variables, which can be substituted by any message or event place name at the base-level during the weaving process. Similarly,

we are using message and event (multi-)term variables  $Ms^v$ , instead of the instantiated specific default (event and message) multi-terms. Thereby, we are boosting the flexibility, by allowing dynamic selection of specific parts from the default behavior. More specifically, instead of a systematic consideration of default initial behavior as indivisible, through such variables we provide the designer with the ability to dynamically select *any part* of that default behavior and combine it with the to-be woven behavior from the aspectual-level (see below).

### 5.4.2 Dynamic-Weaving by Inferring "Non-woven" Rules

We assume thus that any aspect-oriented transition has been behaviorally governed by its corresponding non-woven rewriting rule, through the application of the above definition. Since these non-woven rules still interfere between the aspectual- and the base-level, we require a transformation process to resolve this conflict and result in sort of "woven" transition rules, which can be directly applicable at the base-level using the inherent rewrite theory. Such envisioned transformation process of non-woven rules towards a usual yet emerging base-level rules, should have as objectives: (1) The dynamic selection of any (elements of) ECA-driven rule as-advice from the aspectual-level; (2) The runtime weaving of such selected advices at the base-level, as an emerging yet as standard base-level rewriting rule.

For that purpose, we are introducing a tailored inference rule, that captures the semantics of such aimed transformation of non-woven rules towards woven ones. More precisely, first, we present how to instantiate the read-arc by substituting its aspectual-variables, so that it coincides with a (ECA-driven) token (as-advice) from the aspectual state-rule place. This step is formally captured through the following definition.

**Definition 5.4.2 (ECA-driven rules selection via read-arcs)** We say that the read-arc part of an aspect-oriented transition rule is instantiated by an ECA-driven rule—governing a version  $k$  of a transition  $t(k)$ —from the aspectual-level (i.e. from the rule-state place), if and only if the following conditions are fulfilled.

$$\begin{aligned} \exists \sigma_i : IC_i^v \mapsto [T_{s(p_i)}(X)]_{\oplus}, \quad \exists \sigma_j : CT_j^v \mapsto [T_{s(q_j)}(X)]_{\oplus}, \\ \exists \sigma_c : TC^v \mapsto (T_{StSv}(X) \cup T_{Ms}(X))_{bool} \quad \exists \sigma_p : P^v(-) \mapsto SvP, \end{aligned}$$

With  $p_i$  and  $q_j \in SvP$  and such that:

$$\langle t : k | \otimes_i (\sigma_p(P_i^v), \sigma_i(IC_i^v)), \otimes_j (\sigma_p(Q_j^v), \sigma_j(CT_j^v)), \sigma_c(TC^v) \rangle \in \mathcal{M}(\mathcal{RfP}) \quad (5.1)$$

This definition ensures that for dynamically selecting an ECA-driven rule from the aspectual-level, we should undertake the followings. With respect to a selected aspect-oriented transition  $t$ , we have to find substitutions for input, output and condition parts and associated places, so that the instantiated read-arc matches a concrete governing ECA-driven rule existing at the aspectual-level. More precisely, after applying these substitutions on the read-arc inscriptions, they result in

a token in the marking of the aspectual rule-state place  $R_sP$ . We further note, the substitutions  $\sigma_p$  for the places are to ignored when the respective (message, event and state) places are already instantiated at the concerned read-arc inscription. That is to say, there will be no  $\sigma_p$ , when instead of the place variables  $P_i^v$  and  $Q_i^v$  we are directly using place identifiers such as  $p_i$  and  $q_j$  belonging the (base-level) places set  $S_vP$ .

After demonstrating how to dynamically select any ECA-driven rule from the aspectual-level via read-arcs, we are now ready to present the ultimate step of inferring from the respective instantiated read-arc the right "woven" transition rewriting rule, which can be directly and concurrently applied at the base-level with the other transition rules. The main ideas for such dynamic weaving, consist in propagating the selected (elements of) ECA-driven rule on the different joinpoints, that is, instantiating the aspectual-variables endowing the associated input / output inscriptions and condition of that aspectual transition.

Nevertheless, there are different alternatives for such propagation, depending among others, on the interpretation of the composition operator  $\bowtie$ . That is, as we already discussed this operator can be regarded as conjunction (i.e.  $\wedge$ ), disjunction (i.e.  $\vee$ ) or nil. Besides these variant interpretations of  $\bowtie$ , as we emphasized above, we have to decide which elements of the selected ECA-driven rule should be woven. These include the input inscriptions *and* / *or* output inscriptions *and* / *or* the condition. Furthermore, we have to decide, which specific parts of the default initial transition's behavior have to be dynamically composed with such woven behavior. In the following, we detailed the inference rule capturing such dynamic weaving, corresponding to the case where the composition operator  $\bowtie$  is interpreted as a disjunction  $\vee$  and all the rule elements are woven.

**Definition 5.4.3 (Weaving selected ECA by inferring as rewriting rule)** Let us assume given a simplified "non-woven" as already discussed of the form:

$$\begin{aligned} Tgnr^{nw}(-) : (\mathcal{R}f\mathcal{P}, \langle Tgnr : - \mid (StSv, IC_s^v) \otimes_i (Msi_i, CT_{m_i}^v), \\ (StSv, CT_s^v) \otimes_j (Mso_j, CT_{m_j}^v), TC^v \rangle \parallel_r (\mathbf{StSv}, \oplus_s \langle Si_i \mid prs_s \rangle \bowtie IC_s^v) \otimes_i (Msi_i, ms_i \bowtie IC_{m_i}^v) \\ \Rightarrow (\mathbf{StSv}, \oplus_{s'} \langle Si'_i \mid prs'_{s'} \rangle \bowtie CT_{s'}^v) \otimes_j (Mso_j, ms'_j \bowtie CT_{m_j}^v) \text{ if } TC(Tgnr) \bowtie TC^v(Tgnr)). \end{aligned}$$

We further assume being able to propagate any specific ECA-driven rule from the aspectual-level for this non-woven generic transition rule. That is, the substitutions stated in definition 5.4.2 are fulfilled. More precisely, we have:

$$\begin{aligned} \exists \sigma_s : IC_s^v \mapsto [T_{StSv}(X)]_{\oplus}, \quad \exists \sigma_{mi} : CT_{m_i}^v \mapsto [T_{ms_i}(X)]_{\oplus}, \\ \exists \sigma_{mo} : CT_{m_o}^v \mapsto [T_{ms_o}(X)]_{\oplus}, \quad \exists \sigma_c : TC^v \mapsto (T_{StSv}(X) \cup T_{Ms}(X))_{bool}, \end{aligned} \tag{5.2}$$

Such that:

$$\langle Tgnr : k | (StSv, \sigma_s(IC_s^v)) \otimes_i (Msi_i, \sigma_i(IC_{m_i}^v)), (StSv, \sigma_s(CT_s^v)) \otimes_j (Mso_j, \sigma_o(CT_{m_j}^v)), \sigma_c(TC^v) \rangle \in \mathcal{M}(\mathbf{RsP}) \quad (5.3)$$

By interpreting the composition operator  $\bowtie$  as a disjunction (i.e.  $\wedge$ ), the following inference rule applies and transforms such "non-woven" aspectual rule into woven standard base-level rewriting rule.

$$\frac{(\mathcal{RfP}, \langle Tgnr : k | (StSv, \sigma_s(IC_s^v)) \otimes_i (Msi_i, \sigma_i(IC_{m_i}^v)), (StSv, \sigma_s(CT_s^v)) \otimes_j (Mso_j, \sigma_o(CT_{m_j}^v)), \sigma_c(TC^v) \rangle) \|_r \dots}{Tgnr(k) : (StSv, \sigma_s(IC_s^v)) \otimes_i (Msi_i, \sigma_i(IC_{m_i}^v)) \Rightarrow (StSv, \sigma_s(CT_s^v)) \otimes_j (Mso_j, \sigma_o(CT_{m_j}^v)) \quad \text{if } \sigma_c(TC^v)} \quad (5.4)$$

Please note that because we are interpreting the composition operator  $\bowtie$  as a disjunction (i.e.  $\wedge$ ), the default initial behavior of the transition does not come into play. For that reason, in the inference we are simply ignoring all what come after the read-operator  $\|_r$  by abbreviating all the remaining part of the "non-woven" rule with  $\dots$ .

At this stage, we can keep the default rule and the new emerging one of that transition, or suppress that default one depending on the application at-hand. For instance, we can involve the service state instances in the condition of such rules, and apply both rules but on very specific state instances.

► **Remark 5.4.4** Besides allowing the introduction of completely a new behavior, other variants of that inference rule can be applied to adapt *a specific part* of an existing behaviour. This may concern just the condition by tightening or softening it. We may also choose some specific input (resp. output) arc-inscriptions and enrich them with other knowledge as conjunction or disjunction. More specifically as illustration, if we decide to dynamically tight an existing condition with a shifted-down condition, then the corresponding tailored inference rule will concern just the condition. That is, just the variable  $TC^v$  is to be propagated from the aspectual rule-state place. By ignoring all other components from the read-arc, as we pointed out we are using the symbol ' $\cdot$ '.

This tailored inference rule takes the following form, where the composition symbol  $\bowtie$  is interpreted as conjunction  $\wedge$ ):

$$\begin{aligned} & \exists \sigma \quad TC^v \mapsto (T_{StSv}(X) \cup T_{Ms}(X))_{bool} \\ & (\mathcal{RfP}, \langle t : k | \cdot, \cdot, \sigma_c(TC^v(t)) \rangle) \|_r (\mathbf{StSv}, \oplus \langle Si_i | prs_s \rangle \bowtie IC_s^v) \otimes (Msi_i, ms_i \bowtie IC_{m_i}^v) \\ & \Rightarrow (\mathbf{StSv}, \oplus \langle Si'_i | prs'_i \rangle \bowtie CT_s^v) \otimes (Mso_j, ms'_j \bowtie CT_{m_j}^v) \quad \text{if } TC(t) \bowtie \sigma_c(TC^v(t)) \\ & \hline t(k) : (\mathbf{StSv}, \oplus \langle Si_i | prs_s \rangle) \otimes_i (Msi_i, ms_i) \Rightarrow (\mathbf{StSv}, \oplus \langle Si'_i | prs'_i \rangle) \otimes_j (Mso_j, ms'_j) \quad \text{if } TC(t) \bowtie \sigma_c(TC^v(t)) \end{aligned} \quad (5.5)$$

## 5.5 Aspectual Leveraging for Adapting the CSRV-NETS Flight Service

The purpose of this section concerns the application of the above conceptualization to dynamically adapting the above CSRV-NETS Flight specification. To illustrate this dynamic adaptability, we require to progressively put into play all the above aspect-oriented concepts and mechanisms. More precisely, first we have to endow any transition of this Flight CSRV-NETS specification, with appropriate aspectual-variables, so that they become adaptive-aware. As a second step, we have to build the aspectual-level, through it any ECA-driven 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 on the running slightly upgraded Flight CSRV-NETS specification.

### 5.5.1 Leveraging the CSRV-NETS Flight towards adaptability

As we pointed out, the first step towards endowing any CSRV-NETS service specification consists in preparing that service specification to become adaptable-aware. More precisely, for each of the CSRV-NETS Flight transitions, we have to slightly enrich their (input/output) arc-inscription as well as the condition part with aspectual-variables using the operator  $\bowtie$ .

The resulting of applying this enrichment is depicted in the low-level of Figure 5.7. First note that to ease the manipulation, instead of long names for places (and transitions) we are shortening them. For instance, instead of the place name `Flight_Book`, we are using just `FlBk`. Second, because we want that all business activities of the flight service become adaptable, we are enriching all the three transitions. Again here for sake of simplicity, we are dropping the `Else` part (i.e. the exception cases) in all these transitions.

### 5.5.2 Building and dynamically adapting the flight AOCSRV-NETS

The aim of this step consists in effectively bringing this runtime knowledge-centric adaptability conceptual machinery on such slightly upgraded CSRV-NETS specification. Towards that, we are considering for illustration three emerging business rules scenarios. That is, we are bringing two new rules for the booking request business activity (i.e. the transition `Tflg_rq`) and one rule dealing the canceling activity (ie. the transition `Tflg_c1`). The ultimate goal is to demonstrate that we are able to manipulate any business rule and dynamically shift it down and up on running transitions. These three business rules could informally described as follows:

**Flight to a specific destination (R1)** : This rule says, for instance, that any person traveling to Cairo or Istanbul between June and August gets a discount of 50 percent of the normal fare.

**Flight to a specific destination for group (R2)** : Any two persons traveling for instance to Paris during the month of December will automatically get 30 percent discount.

**Seasonial 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 the half of the paid price.

The next step after this informal description, of any emerging or existing ECA-driven business rule to be dynamically integrated in the running Flight CSRV-NETS specification, consists in *formally* expressing it as a five-element advice with respect to the transition governing the associated business activity. In the following, we translate these three informal rules to their precise five-element tuples description.

**The business rule R1 as advice.** 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 Flight CSRV-NETS specification. As it is the first rule to be introduced, besides the default initial rule, the counter for the version is to be set to one (1). Moreover, when analyzing this business rule, we see that 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 behavior (initial business rule). As we afore-suggested, in this case the second and third elements in the advice-as-tuple have to be set to ”-”. Finally, using the aspectual variables from the (default) input messages and the input service state the described conditions are straightforwardly expressed into the following formal expression:  $(Fr = \text{"Cairo"} \vee \text{"Istanbul"}) \wedge (\text{"June"} \leq Dt \leq \text{"August"}) \wedge (Py := Cx * .50)$

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

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

**The business rule R2 as advice.** 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 behavior, 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 `F1Rq` (flight-request) we require two messages, one for instance from `Cs1` and the second from `Cs2` with similar 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, to be formulated as for the above rule. We should just note that since two persons are at stake, the available seats should be greater than 2 and both the two customers Ids (i.e. `Cs1` and `Cs2`) have to be added to the reservation list `Rs`. The flight

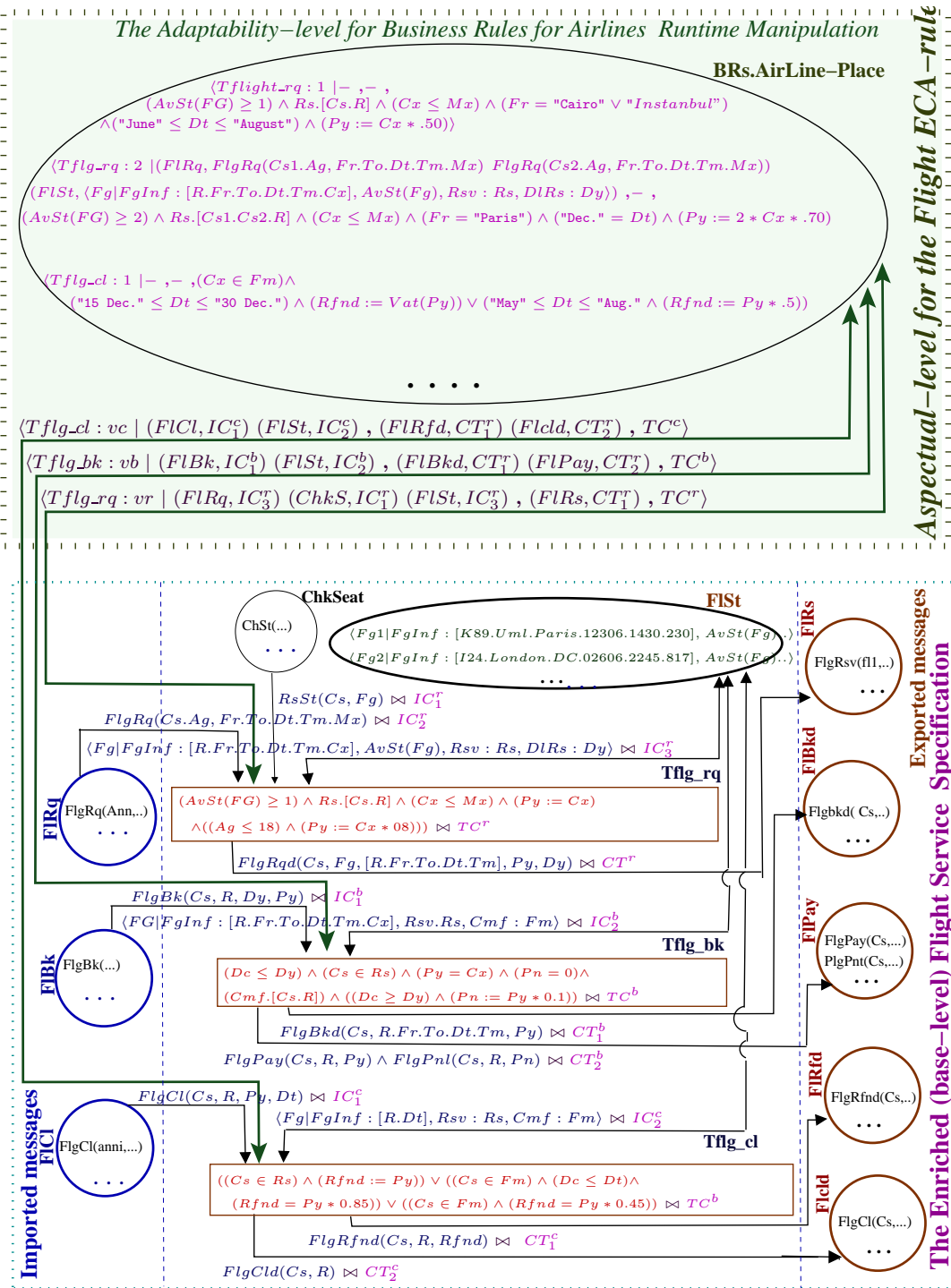


Figure 5.7: The Runtime Adaptable AOCSRV-NETS flight service before rules weaving



cost should of course be less than the max budget of any of the two customers. All in all this tuple takes the following form:

$$\langle Tflg\_rq : 2 \mid (FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx) \oplus FlgRq(Cs2.Ag, Fr.To.Dt.Tm.Mx)) \\ (FlSt, \langle Fg \mid FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy) \rangle, -, (AvSt(FG) \geq 2) \wedge \\ Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx) \wedge (Fr = \text{"Paris"}) \wedge (\text{"Dec."} = Dt) \wedge (Py := 2 * Cx * .70) \rangle$$

**The business rule R3 as advice.** This new business rule concerns the cancel activity and thus the transition  $Tflg\_cl$  as first version besides the default one. As for the first introduced rule, this rule mainly focusses on the condition part, and henceforth the two input and output second and three elements of the tuple are abbreviated to ”-”. The condition itself is composed of a disjunction of two expressions: One concerning the Christmas period (i.e. between 25 and 30th of December), where the sum to be refunded is to set the VAT, and the summer period where just the half is refunded. All in all the resulted tuple is to be expressed as follows:

$$\langle Tflg\_cl : 1 \mid -, -, (Cx \in Fm) \wedge (\text{"25 Dec."} \leq Dt \leq \text{"30 Dec."}) \wedge (Rfnd := Vat(Py)) \\ \vee (\text{"May"} \leq Dt \leq \text{"Aug."} \wedge (Rfnd := Py * .5)) \rangle$$

### 5.5.3 Emerging the rules-as-advicees at the aspectual-level

As depicted in Figure 5.7, 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 aspectual-level transition AD2RL (for the second rule RLn2AD as it is the second version). That means that the place Ad2R1 should have been containing three tokens of the form:

- $Add\_Bh(Tflg\_rq, -, -, (Fr = \text{"Cairo"} \vee \text{"Instanbul"}) \wedge (\text{"June"} \leq Dt \leq \text{"August"}) \wedge (Py := Cx * .50))$
- $Add\_Bh(Tflg\_rq, (FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx) \oplus FlgRq(Cs2.Ag, Fr.To.Dt.Tm.Mx))(FlSt, \langle Fg \mid FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy) \rangle, -, (AvSt(FG) \geq 2) \wedge Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx) \wedge (Fr = \text{"Paris"}) \wedge (\text{"Dec."} = Dt) \wedge (Py := 2 * Cx * .70))$
- $Add\_Bh(Tflg\_cl, -, -, (Cx \in Fm) \wedge (\text{"15 Dec."} \leq Dt \leq \text{"30 Dec."}) \wedge (Rfnd := Vat(Py)) \vee (\text{"May"} \leq Dt \leq \text{"Aug."} \wedge (Rfnd := Py * .5)))$

The firing of the aspectual-level transition AD2RL three times successively results in the emerging of the three tuples in the rule-place  $Brs.Airline\text{-}Place$  as depicted in the upper-layer of Figure 5.7. That is to say, three new rules have been dynamically emerged at that adaptability-level. To keep the Figure manageable we have indeed skipped this self-explained firing. Giving these rules, it is important to emphasize that we can in the same spirit change and / or delete them through the two other aspectual-level transitions and their places.

### 5.5.4 Runtime shifting down / up of rules-as-advice on the Flight CSRV-NETS service

After demonstrating how any business rule can be manipulated at runtime at the adaptability-level, we are ready to (un)weave any of these emerging rules, through the proposed inference rules. That is, two-step based firing for read-arcs is necessary. Furthermore, we can decide for any strategy for weaving such rules, that is, unweave the old running rules and / or combine the new behavior with that default one, and so on. In the following, we illustrate this runtime adaptability on default behaviors for both the flight request and cancel transitions. More precisely, let us bring down the rule (R1) as default behavior for the transition `Tflg_rq` and the rule (R3) as default behavior for the transition `Tflg_cl`. This also means that the default behaviors 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-down consists in introducing the two emerging behaviors, through the weaving inference rule we explained. We are skipping detailing that straightforward application of these inference rules, and assuming directly that the request-transition default behavior has been shifted-up and these new emerging rules have to be shifted-down. The ultimate resulting is depicted in Figure 5.8. That is, first the default behavior for both transitions `Tflg_rq` and `Tflg_cl` is shifted-up at the adaptability-level. Secondly, the behavior 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 position to dynamically evolve any CSRV-NETS specification and monitor the evolution.

For instance, by instantiating the weaving inference rule on the first emerging rule for the request R1, we result in the corresponding new rewriting rule:

$$\begin{aligned}
 \mathbf{Tflg\_rq(2)} : & (FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx) \oplus FlgRq(Cs2.Ag, Fr.To.Dt.Tm.Mx)) \otimes \\
 & (FlgSt, \langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, DlRs : Dy \rangle) \\
 & \Rightarrow (FLGRQD, FlgRqd(Cs, Fg, [R.Fr.To.Dt.Tm], Py, Dy)) \\
 & \text{if } (Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx) \wedge (Fr = \text{"Paris"}) \wedge (\text{"Dec."} = Dt) \wedge (Py := 2 * Cx * .70))
 \end{aligned}$$

At the same time, by adopting a disjunction semantics for the operator  $\otimes$ , we are disabling the default request transition behavior. To reflect that, we need to unweave or shift up to the aspectual-level this default behavior. To achieve this shifting up, we have just the apply the weaving inference rule in the reversible sense. In other words, the premise becomes the rule conclusion and the conclusion corresponds to the adding of that default rule as tuple to the aspectual rule-place.

## 5.6 An Aspect-oriented MAUDE for Validating AOCSRV-NETS

In the previous chapter, we demonstrated how to leverage the MAUDE language, so that it can concurrently validate and reason about CSRV-NETS specifications. In particular, we introduced the concept of MAUDE-based service component. More specifically, capitalizing on CSRV-NETS features, we accordingly tailored MAUDE (object-)configuration towards component-configuration. We

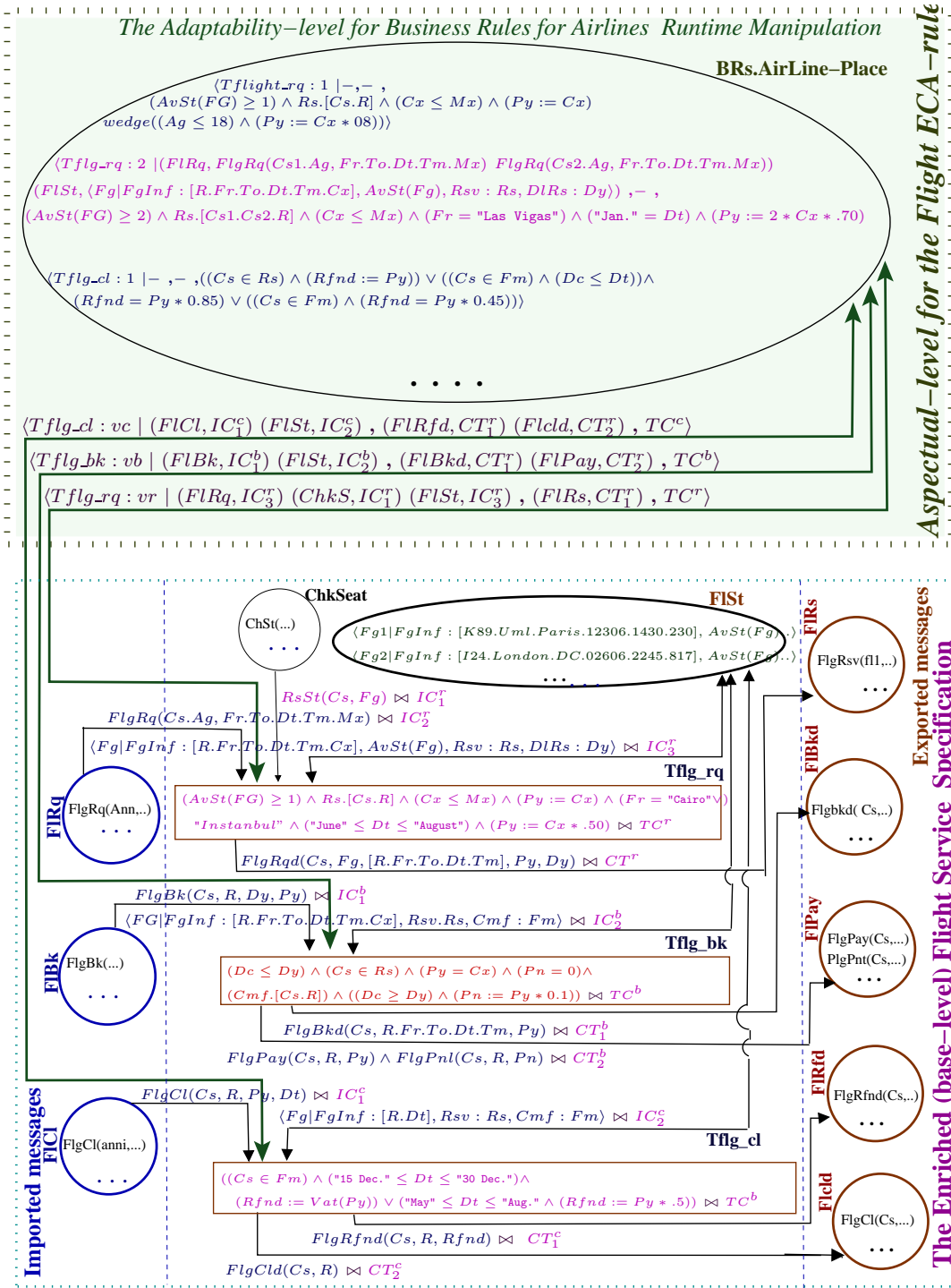


Figure 5.8: The Runtime Adaptable AOCSRV-NETS flight service after rules weaving

then applied this MAUDE-based service componentization on the CSRV-NETS Flight specification. Furthermore, in the previous sections of this chapter, we presented an aspect-oriented leveraging of this formalism, we referred to as AOCSR-NETS, to handle runtime adaptability.

The purpose of this section consists thus in pushing the proposed MAUDE extension one step further, so that it allows semantically governing and reasoning about AOCSR-NETS—instead of just CSRV-NETS specification and graphical animation. More specifically, recapitulating on AOCSR-NETS aspect-orientation, we have to empower MAUDE service components with aspect-oriented features, so they become dynamically adaptable.

A straightforward and naive alternative to semantically interpret AOCSR-NETS in rewriting logic and MAUDE consists of the following steps. First, we have to translate the (aspectual-level) rewrite theory of the aspectual Net into MAUDE. Then, with respect to any specific aspectual Net, we interpret its transitions as rewriting rules. Secondly, for dynamically (un-)weaving any ECA-driven rule, we have to express the associated inference rules, as deduction rules in MAUDE. Finally, we should upgrade any base-level rewriting rules, with aspectual-variables as joinpoints.

Nevertheless, such direct MAUDE-based operational semantics for AOCSR-NETS may suffer from several severe limitations. First, since the rewrite theory of the aspectual-level is little-bit complex (with aspectual variables and non-instantiated states), the translation as well as the reasoning require deep expertise in MAUDE. The same difficulties applied on the interpretation and working on the inference rules regulating the shifting up and down. Third, with such straightforward interpretation, we are not exploiting all the capabilities of MAUDE, and more particularly the reflection-level and its advanced computational and shifting up / down primitives [CM96, CDE<sup>+</sup>07].

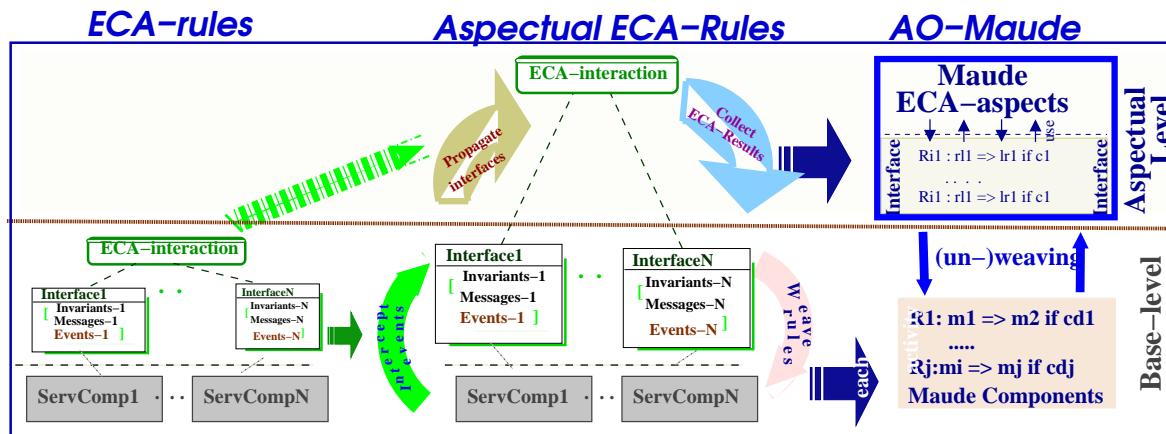


Figure 5.9: From ECA-driven service interactions into a Compliant Aspect-oriented MAUDE extension for AOCSR-NETS

More precisely, towards overcoming all shortcomings of such direct interpretation, we propose a more lightweight semantics by exploiting all the MAUDE capabilities. More precisely, as depicted in Figure 5.9, the main ideas consists in abstracting from the AOCSR-NETS tedious concepts.

That is, instead of directly handling AOCSRV-NETS base- and aspectual-levels, we aim focussing exclusively on the governing architectural ECA-driven rules as main driving for dynamic adaptability. More precisely, inspired by the AOCSRV-NETS aspectual-level, we propose how to endow any architectural ECA-driven rules with aspectual-oriented mechanisms, so that it becomes dynamically adaptable [ABS09c]. Then, capitalizing on MAUDE reflection capabilities, we faithfully capture this aspect-oriented architectural ECA-driven rules by accordingly leveraging MAUDE service components.

Generally speaking, the envisioned aspect-oriented leveraging of architectural ECA-driven rules and its compliant MAUDE-based foundation, could be summarized through the following steps. First, we should be able to intercept any triggering events targeting service components. Second, such interception should lead to extraction of all required features, via respective service interfaces to be propagated to the aspectual-level. Third, at this aspectual-level, the right ECA-driven rule(s) has to be selected and performed on (and only on) the invoked participant instances. Fourth, the resulting outputs of the performed rule(s) should dynamically woven on running service components, via their interfaces.

Following these motivations, the rest of this section is organized as follows. First, we present how to "abstractly" and gradually endow any architectural ECA-driven rule with aspect-oriented mechanisms. Second, we present how besides MAUDE-based service components, service interfaces with extra-weaving capabilities are to be specified. Third, we present how to capture ECA-driven rules by extending MAUDE. Finally, we present how the dynamic weaving is captured with the MAUDE extensions and its reflection capabilities.

For the rest of this section, instead of the non-trivial flight service, we adopt a more simplified illustration based on a banking withdrawal process. Indeed, to stay competitive, banking systems are offering different incitive packages for their customers. These packages range from basic agreed-on contracts (e.g. different formulas for withdrawal / transfer moneys) to sophisticated complex offers (i.e. staged housing loans, mortgages, etc.) depending on customers profiles (e.g. assets, trust, experiences, etc). So, even for a basic withdrawal it is not acceptable to hide its logic inside entities such as customer or account. A withdrawal must instead be regarded as an *agreement* between the customer and his/her account(s). This directly leads to more transparent and flexible tailored withdrawal.

As illustration, we propose two exogenous withdrawal agreements. The first basic withdrawal consists simply in externalizing the withdrawal constraint (i.e.  $\text{balance} > \text{amount}$ ) from the account. In this manner, it can be adapted as the customer wishes. We may speak then of withdrawal with credits (i.e. Crd-withd.), where a specific credit is given to the customer.

```
ECA-Interaction Crd-Withdraw
participants Acnt: Account;
               Cust: Customer
attribute credit : Money
invariants Cust.own(Acnt) = True
```

```

interaction rule : VIP
at-trigger Cust.withdraw(M)
  under Acnt.bal() + credit ≥ M)
  acting Acnt.Debit(M)
end Crd-Withdraw

```

We should note that due to their simplicity all required interfaces from the customers and accounts are skipped here. For instance, for the standard withdrawal (resp. credit one), the customer should provide the withdraw event (and the credit amount), whereas the account should provide the balance (shortened as `bal`) and the Debit method.

### 5.6.1 Aspect-orientation of architectural ECA-driven rules for Dynamic Adaptability

As depicted in the left-hand side of Figure 5.9, we assume given an architectural ECA-driven service interaction, putting several service partners into cooperation. At the instance level, this cooperation brings into contract specific partner instances (through their service interfaces). For instance, we may have customer such as *Cs1* under standard withdrawal with his / her account *Ac1*, whereas customer *Cs2* under CRD-withdrawal interaction with the account *Ac2*. For such interaction instances and only for them<sup>3</sup>, we propose to intercept their triggering events as well as any required properties. The running interaction glues are then executed at that interaction-level. Finally, the resulting emerging actions and states are to be dynamically woven on respective service components. With the aim to facilitate any subsequent operational foundation, we have thus abstracted these ideas as given in middle of Figure 5.9. These five steps can be highlighted as follows.

1. **Events interception and properties extraction** : The first step we propose towards aspect-orientation consists in intercepting any triggering events / messages from directly going to the service component. On the basis of the triggered instances, the required properties for the ECA-driven interaction are extracted from participating services to build the interfaces.
2. **Features propagation to the aspectual-level** : The next step aims at shifting up these required interface instances to the ECA-driven aspectual-level, where the respective rules are residing.
3. **Execution of right interaction rules**: In terms of aspect-oriented mechanisms, the corresponding ECA-driven rule(s) are to be performed as cross-cutting (agreed-on) advices.

---

<sup>3</sup>We avoid intercepting every event but just those relevant in the current interactions state.

4. **Propagation of resulting behavior** : This step consists in dispatching the output of the performed ECA-driven rule on respective interfaces.
5. **Dynamic weaving on running components**: This final step concerns the weaving and execution of the corresponding actions on respective service components. This weaving should be non-intrusive on the running service components, that is, without such service components being aware of it (i.e. as if no-interception has been applied).

### 5.6.2 Towards an ECA-Compliant aspect-orientation of MAUDE

Towards formally validating the above conceptual aspect-orientation of ECA-driven rules for service runtime interaction and dynamic adaptability, we propose to leverage the already extended MAUDE with service components towards a compliant aspect-orientation. Towards that purpose, we first present how service interfaces can be conceived from associated MAUDE-based service components. Second, we develop on how to express any ECA-driven rules as a MAUDE-based advice. Finally, capitalizing on MAUDE reflection capabilities, we present how to dynamically (un)weave such ECA-driven rules on running service components.

#### Service interfaces in the extended MAUDE

In contract to MAUDE service components, associated service interfaces should deal only with observed features (i.e. observed properties, events and messages). Furthermore, to keep track of different instantiation of a given service interface we require unique identification. Last but not least, since such service interfaces are aimed to propagate information to the aspectual-level as well as receiving results, they have to be endowed with extra mechanisms for that purpose.

Taking these observations into account, first, we propose as service interface structure the following: `[IntfIdentif|Interface-conf.]`. That is, any service interface (state) is composed of unique identification and a current configuration composed of observed events, messages and properties needed for the aspectual-level. Furthermore, we are anticipating how we should intercept events and properties and propagate (resp. receive) them to the aspectual-level. The aim is to capture the first step from the five-steps, we previously detailed. As detailed below, the interception rule `intercepts` and extracts from the service component configuration any part that interests the ECA-driven interaction. These can be events and properties. Through the rule `Subsume` in line 24, the supposedly intercepted events and/or properties are mapped to the specific structure of service interfaces. The weaving rule `weaveCfIntf` instead permits enriching respective service component, with the resulting actions (after the interaction being executed).

```

1. mod INTF_GNR is
2. inc CMP_GNR ...
11. op [_ | _] : Intf_NM ConfINTF → EX_ConfIntf .
13. op subsume(.,_) : ConfINTF Intf_NM → E_ConfIntf .

```

```

14. op belong(.,_) : Iid IidL → Bool .
15. op weaveCfIntf(.,_) : EX_ConfIntf ConfCMP → ConfCMP .
16. op intercept(.,_):ConfINTF ConfCMP → ConfCMP ...
24. rl [Subsume]:subsum(Cfintf, INM)⇒[INM | Cfintf] .
25. rl [weaveCfIntf]:weaveCfIntf([INM|Cfintf],Cfcmp)
    ⇒Cfintf Cfcmp .
26. rl [intercept]:intercept(Cfintf,Cfintf Cfcmp) ⇒Cfcmp .

```

► **Example 5.6.1** To illustrate such MAUDE service components and interfaces, we re-consider the account withdrawal case. That is, for the account service component, we assume that the balance `bal` is observed, whereas the limit `limt` is local. Similarly, we propose that the *credit* and *debit* be observed messages, whereas the change-of-limit be a local one (lines 12-15). Please note that, the debit rule now contains *no conditions*. We are thus externalizing any business logic at the interaction aspectual-level. So, the conditions will be later evolved and woven as aspects on the (basic) service component rules.

```

1. mod ACNT_CMP is ...
10.op bal: _ : Rat → obs_Prop .
11.op limt: _ : Rat → loc_Prop .
12.op Crd(.,_) : AcntId Rat → CRDT .
13.op Db(.,_) : AcntId Rat → DBT .
14.op ChgL(.,_) : AcntId Rat → CHGL ...
22.rl [credit] : Crd(A, M) < A|bal : B >⇒< A|bal : B + M > .
23.rl [debit] : Db(A, M) < A|bal : B >⇒< A|bal : B - M > .
24.rl [chgl] : ChgL(A, L1) < A|limt : L >⇒< A|limt : L1 > .

```

The account service interface to be involved in the withdrawal ECA-driven rule is depicted below. We thus require the debit and the balance properties. Moreover, we have to intercept the balance (rules `getCfIntfbal` and `getCfIntfbalf`).

```

1. mod ACNT_INTF4WDR_GNR is
8. op ACNT : → Intf_NM .
9. op Db(.,_) : AcntId Rat → DB .
10. op bal: _ : Rat → obs_Prop [ctor gather (&)] .
18. rl [getCfIntfbal] : getCfIntfbal(<AC|bal:B > Cfcp, AcntsL)
19.   ⇒ (if belong(AC, AcntsL)
20.   then < AC|bal:B > getCfIntfbal(Cfcp, AcntsL)
21.   else getCfIntfbal(Cfcp, AcntsL) fi) .
22. rl [getCfIntfbalf] : getCfIntfbalf(Cfcpf
23.   getCfIntfbal(Cfcp, AcntsL)) ⇒ Cfcpf .

```



### ECA-driven rules as MAUDE-based advices

So far we leveraged MAUDE configuration to intrinsically support service components and interfaces. We also anticipate how intercepting and weaving required properties and messages / events. Based on that, we now present a MAUDE formalization and execution of the ECA-driven interactions as aspect-oriented advices. First, we define a MAUDE-based algebraic structure to capture all elements, from any ECA-driven interaction. These elements are composed of: (1) an ECA-driven interaction name (e.g. `WdrStd` or `WdrViP`); (2) identifiers for participating interfaces (e.g. `ACNT` and `CUST`); (3) any specific information, such as properties and operations, we may require besides those from the interfaces (e.g. the credit attribute). We accordingly propose the structure for any ECA-driven rule as a tuple:

$$[CoorName \parallel (partner\_ids(\$partner\_ids)^*)@coord\_infos] (\&[Partner_i;partner\_infos])^+$$

The MAUDE-based formalization of this ECA-driven aspectual interactions is given as follows. To exhibit a maximum of concurrency, we allow different parts of that term to be split and recombined. This split / recombine capabilities are captured using two respective rules as given in that specification, namely (`Split_CfIntf` and `Recombin_CfIntf`). Furthermore, to prepare superposition of the resulting interaction on different components, we permit the extraction of any part using the rule `extractCfIntf` (line 27).

```

1. mod ASP_COORD is
2. inc INTF_GNR ...
15. op _ $ _ : PartnerIds PartnerIds → PartnerIds [ac] .
16. op _ ; _ : Attributes Attributes → Attributes [ac] .
17. op _ . _ : CoordOpers CoordOpers → CoordOpers [ac] .
18. op _ @ _ : PartnerIds Attributes → PartnerAttrs .
19. op [ _ || _ ] : Coord_NM PartnerAttrs → ObjCoord .
20. op extractCfIntf( _ , _ ) : ConfCoord Intf_NM → EX_ConfIntf .
21. op _ & _ : ConfCord ConfCord → ConfCord [ac] ...
25. rl [Split_CfIntf] : [Inm | CfI1 CfI2]
26.     ⇒ [Inm | CfI1] & [Inm | CfI2] .
27. rl [Recombin_CfIntf] : [Inm | CfI1] & [Inm | CfI2]
28.     ⇒ [Inm | CfI1 CfI2] .
29. rl [extractCfIntf] : extractCfIntf([Inm | CfI1]
30.     & CfCt, Inm) ⇒ [Inm | CfI1] .

```

► **Example 5.6.2** We report on the Credit-withdrawal interaction rule (`WrdVip`). We first define the specific attribute `Credit` (shortly as `crd`). We have chosen `CS` for the customer and `AC` for the account as identifiers. The Credit-withdrawal rule says that when a withdrawal event is sent from a customer partner `CS`, it is intercepted via the customer service interface `CUST`. It then enters in contact with the balance from an agreed-on account partner `AC`, provided via the service interface `ACNT`. The right-hand side says that under a specific credit `crd` and condition on the balance, this

interaction results in: (1) the withdrawal event being consumed (i.e.  $[CUST|nil]$ ); and (2) sending of a debit message to the account (i.e.  $[ACNT| < AC|bal : B > Db(AC, M)]$ ).

```

1. mod COORD_WdrVip is ...
6. op WdrVip : → Coord_NM .
7. op crd: _ : Int → Attribute [ctor gather (&)] ...
11. crl [WdrVip] : [WdrVip || (CS $ AC) @ crd: C]
12. & [CUST | Wdr(CS, M)] & [ACNT | < AC|bal:B >]
13. ⇒ [WdrVip || (CS $ AC) @ crd: C]
14. & [ ACNT | < AC | bal: B > Db(AC, M)]
15. & [CUST | nul] if (B + C) ≥ M .

```

### 5.6.3 Dynamic (un)weaving of aspectual MAUDE service-interactions

So-far we presented how to specify MAUDE-based service components, interfaces and ECA-driven interactions as-advice. The last step towards the strived non-intrusive dynamic adaptability consists in judiciously composing them. More precisely, towards non-intrusively intercepting events, executing the rules and then weaving respective ECA-driven aspectual interactions, the generic guidelines for any strategy should respect the following steps. First, we propose to split any involved service component configuration and prepare it for intercepting any events and required properties. Second, we propose to intercept only those instances in agreements at the interaction aspectual-level. Third, we have to propagate these intercepted service interface states to the aspectual-level. Fourth, we perform the interaction rules as-advice on these service interface states. Finally, we have to extract all resulting interface states and weave them on respective running service components.

As we already emphasized, in terms of aspect-oriented mechanisms, the ECA-driven interaction behaviors are playing the role of (cross-cutting) advices. The reflection strategy itself represents the pointcuts, that is, how to (intercept and) weave the advices. Finally the jointpoints represent the rules at the service component-level, which are non-intrusively enriched with such ECA-driven behaviors. For instance, the debit method is externally enriched in our case with the balance sufficiency (plus the credit).

► **Example 5.6.3** The Credit-withdrawal strategy below concretely reflects the above steps. First the `SplitAT` rule is applied to the Account and Customer service component configurations. Then, using the `belong` rule, we extract the state parts in agreements. Thirdly, through the rules `getCfIntfwdr` and `getCfIntfwdrf`, we intercept all withdraw events from the customer in Credit-withdrawal agreement. These service interface states are adapted to the interaction structure using `Subsume`. The Credit-withdrawal interaction rule is then performed using (`WdrVip`). The rules `extractCfIntf` and `weaveCfIntf` permit finally to weave the results on the account service component.

```

1. mod ASP_WDR_Str is ...
11.op Compute : Term Nat → Term .
12.ceq Compute(T, N)

```

```

13.= if(N == 1) then
14.(if(SplitAT? :: Result4Tuple)
15.then Compute(getTerm(SplitAT?), N)
17.then Compute(getTerm(belong?),N)
21.then Compute(getTerm(getCfIntfwdr?),N)
25.then Compute(getTerm(getCfIntfbal?),N)
27.then Compute(getTerm(intercept?),N)
31.then Compute(getTerm(Split_CfIntf?),N)
33.then Compute(getTerm(WdrVip?), N)
36.then Compute(getTerm(Recombin_CfIntf?),0)
38.then Compute(getTerm(extractCfIntf?), 0)
40.then Compute(getTerm(debit?))

```

Figure 5.10 depicts the application of the above strategy on concrete account and customer service component configurations. Note that both standard and Credit-withdrawal rules come into play, as specific customers and accounts are in agreements with respect to both. That is, for instance here **Manru** is in **Std** with the account and **ManruAC**, whereas **Nas** is as **Credit** contract with the account **NasAC** (with 500 as credit).

The figure shows a Maude development environment with three code windows and one emulator window.

**ACNT\_CMP\_CONF\_GNR.maude**

```

mod ACNT_CMP_CONF is
  ex ACNT_CMP .
  ops AC1 AC2 AC3 AC4 NasAC ManruAC :-> AcntId .
  op AcCfCp :-> AcntCf .

  eq AcCfCp = < AC1 | bal: 300, limt: 5 > Crd(AC1, 30) Db(AC2, 50)
             < AC2 | bal: 800 > Trs(AC1, AC3, 80) < AC3 | bal: 150, limt: 50 >
             ChgL(AC4, 10) < AC4 | bal: 550, limt: 20 > Crd(NasAC, 110)
             < NasAC | bal: 1500, limt: 50 > Crd(ManruAC, 90)
             < ManruAC | bal: 200, limt: 5 > .
endm

```

**CUS\_CMP\_CONF\_GNR.maude**

```

mod CUS_CMP_CONF is
  ex CUS_CMP .
  op A1 A2 A3 A4 Nas Manru :-> CustId .
  op CsCfCp :-> CustCf .

  eq CsCfCp = Wdr(A1, 25) < A1 | Adr: "Mag." > Wdr(A2, 40)
             < A2 | Adr: "Berlin" > Wdr(A3, 60) < A3 | Adr: "Frank" >
             Wdr(A4, 95) < A4 | Adr: "Peking" > Wdr(Nas, 35)
             < Nas | Adr: "Algeria" > Wdr(Manru, 75)
             < Manru | Adr: "Zhengzhou" > ChgAdr(Manru, "Luoyang") .
endm

```

**ASP\_COORD4StdVip\_CONF\_GNR.maude**

```

mod ASP_COORD_CONF is
  ex COORD_WdrStdVip .
  inc CUS_INTF_CONF_UP .
  inc ACNT_INTF_CONF_UP .
  inc ASP_COORD_Part .

  eq CCF = [WdrStd | Manru $ ManruAC] &
           [WdrVip | (Nas $ NasAC) @ crd: 500] &
           CfintrCus & CfintrAcnt .

  eq ACIntCF = extractCfintr(CCF, ACNT) .
  eq CSIntCF = extractCfintr(CCF, CUST) .
endm

```

**Emulator Result**

```

reduce in ASP_WDR_Str: downTerm(Compute(upTerm(AcCfCpDw), 1), 'error) .
rewrites: 3159 in -422050321716ms cpu (2934ms real) (~ rewrites/second)
result ConfCMP: < ManruAC | bal: 125 > < ManruAC | limt: 5 > < NasAC | bal:
1465 > < NasAC | limt: 50 > < AC1 | bal: 300 > < AC1 | limt: 5 > < AC2 |
bal: 750 > < AC3 | bal: 150 > < AC3 | limt: 50 > < AC4 | bal: 550 > < AC4 |
limt: 20 > Crd(ManruAC,90) Crd(NasAC,110) Crd(AC1,30) ChgL(AC4,10) Trs(AC1,
AC3,80)
=====

```

Figure 5.10: Dynamic weaving of Std- and Crd-withdraw interaction rules

## 5.7 Towards a compliant .NET environment WS-deploying of AOCSRV-NETS

As we already emphasized, current Web-Services standards such as WSDL and BPEL are purely process-centric and manual. Consequently, directly adopting them for implementing the proposed approach simply means losing all the strengths we were striving for, namely dynamic adaptability, rule-centricity and separation of concerns.

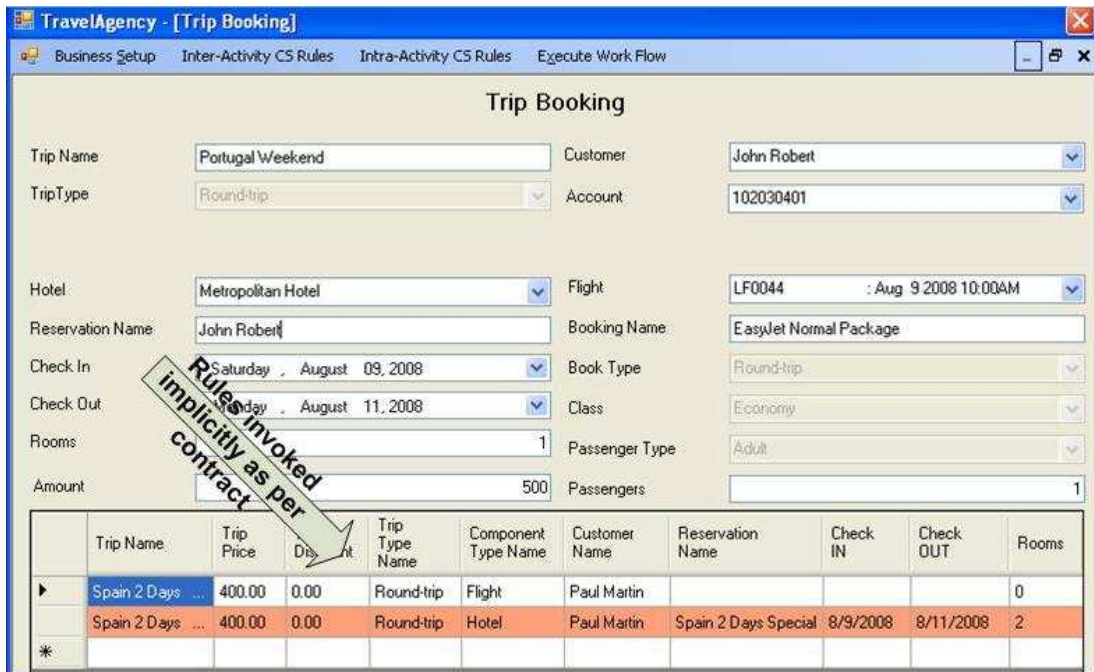


Figure 5.11: The IDE Environment and its main functionalities

We are thus instead proposing the .Net [AW02] environment and its recent extensions with aspect techniques [WFF]. The .NET framework supports syntax for multiple programming languages including C#, VB.NET, J# and C++ all generating Common Intermediate Language (CIL) on compilation. The .NET platform provides excellent support for software extensibility, adaptability, maintainability and customizability through various techniques including reflection, proxy, intermediate language and on-fly code generation.

By fully exploiting these .NET capabilities we are proposing a compliant service-oriented implementation of the approach. More precisely, the main IDE of the .NET supporting tool we implemented inherently reflects the forwarded founded stepwise approach to runtime adaptive and rule-centric service-oriented business processes development. In this main IDE exposing the implemented environment in Figure 5.11, first under the icon "business process", business process reference / name can be introduced, deleted or updated. The second icon "business activity" allows for manipulating the business activities composing a given business process. As we motivated

above, the partial-ordering of such activities within a given business process are postponed to late stages. For instance, depending on the business rules in place, governing a given business process, which are taken in charge thorough the third, fourth and fifth icons, different semantically-driven profiled ordering can be put into place. More precisely, the third icon "Add/remove CS rule" allows for editing, creating or deleting any ECA-driven rule with respect to given business activity within a specific business process.

Any of the implemented rules can be instantiated as first-class independent entities using the fourth icon "CS Rule Instantiation". As we are striving for dynamically changing any existing business rule, we implemented a main functionality for doing so through the icon "Adapt CS rule". Finally, the last functionality allows for defining a concrete workflow (from the general business process) and executing any case.

### 5.7.1 Mapping and manipulation of Conceptual ECA in .NET

To facilitate a smooth yet conservative mapping of the conceptual ECA-driven aspectual-level to the .NET service-based implementation, we have been benefiting from the extensive capabilities of the using Microsoft Work Flow Foundation (WFF) [AW02, WFF]. Indeed, WFF supports, among others, the manipulation of (event-driven) business rule using a suitable XML-based templates. It further allows for defining, managing and executing stateful workflow. WFF consists thus of an activity model, workflow designer and XML-based rules engine. Rule-Definitions tag is composed of multiple RuleSets. Each RuleSet is associated to a given business activity. A RuleSet is a collection of rules, where each RuleSet is composed of Rule. Rule contains the ECA specification as Then-Actions and Rule.

Figures 5.12 and 5.13 recapitulate the main translating steps of the ECA-driven conceptual level towards the developed .NET environment. Firstly, as it should be expected different participating business entities are internally implemented as Web-Services. The exposed functionalities, we require to achieve any given interaction are of course captured as Web-Service interfaces (described using WSDL). The second and main translation concerns the mapping of the ECA-driven rules themselves. First, we capture the rule itself as a (composite) Web-Service, which can be mostly owned by any of the involved providers; though we also consider the case of third-party ownership. For instance, in our application the rules are normally owned by the bank; but they can be outsourced to a third-party for more optimal, universal and intelligent management. Second, the ECA-rule for a given business activity are conceived as a workflow. In this ECA-driven intra-activity workflow, instantiations of the rules can be performed. Since, each rule is implemented as an aspect, the three elements (event-condition-action) composing a rule can be woven on the basis of the instantiated rules.

It is worth mentioning at the end that, in [RLA+09, URAS09] we experienced the environment with both case-studies. We further developed an advanced database-driven mechanisms for enhancing the persistency and conversational-level of different intra- and inter-ECA-driven interac-

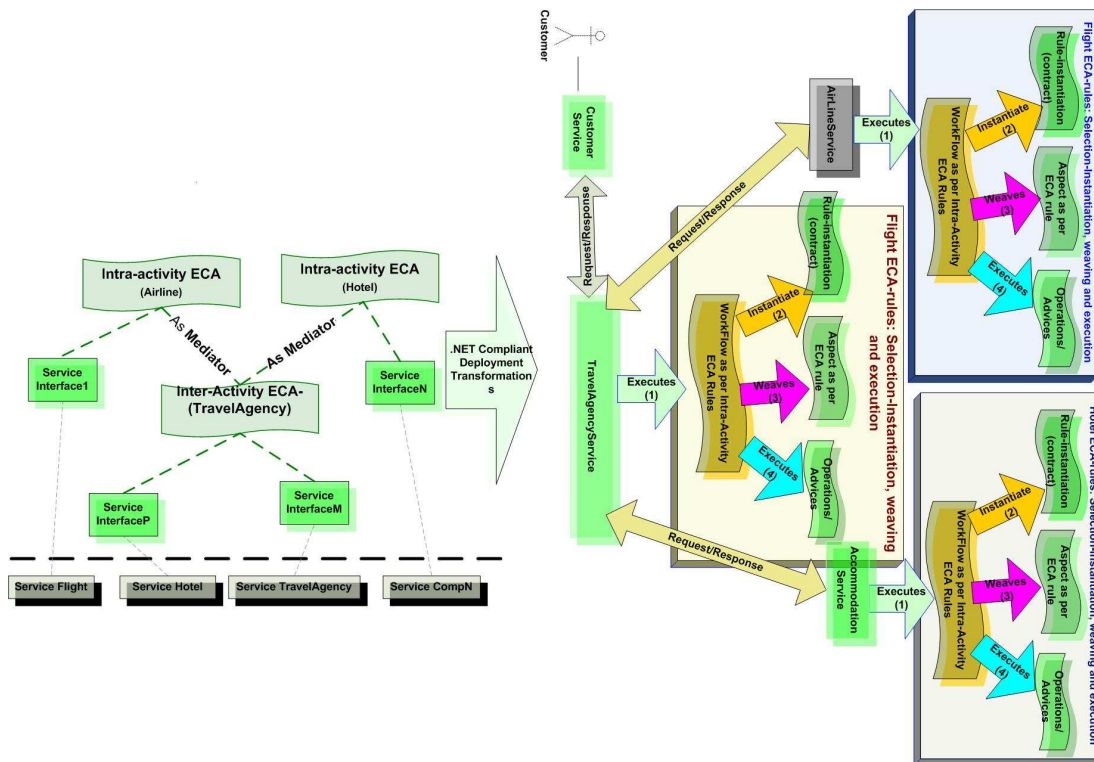


Figure 5.12: The principles of mapping ECA-centric rules to the Aspectual .Net Env.

tion rules. The automatic interconnection between the forwarded aspect-oriented MAUDE and the environment is ongoing, by exploiting the moment project, which allows running MAUDE directly under JAVA environment.

## 5.8 Chapter Summary

Along this chapter, we tackled the crucial and challenging problem of developing dynamically adaptable service-oriented applications in a stepwise and disciplined manner. We first focussed on the stepwise leveraging of CSR-V-NETS service specifications towards an aspect-oriented two-level

Aspectual ECA-Conceptual Level	AOP .NET mechanisms
Business entities	Service components
Entities interfaces	WSDL-based interfaces
ECA-rules	WFF-based Rulesworkflow
ECA-rule dynamic selection	WFF-based workflow with AOP advices

Figure 5.13: Translating steps from ECA-conceptual to the compliant .NET env.

based formalism. At the aspectual-level, we proposed how ECA-driven rules can be dynamically manipulated as advices. At the conceptual-level, we smoothly enriched CSRV-NETS service components with joinpoints, we associate to any to-be adaptable transition. Through read-arcs, we then related the two-levels. Finally, using tailored inference rules as pointcuts, we demonstrated how to dynamically weave and unweave any (specific elements of) emerging ECA-driven rule. We further illustrated this incremental approach to runtime adaptable services using the already conceived Flight service. For formal validation and reasoning, the chapter further proposed a tailored aspect-oriented version to the MAUDE language, that is compliant with the forwarded aspect-oriented AOCSRV-NETS conceptualization. Finally, we also addressed the efficient Web-Services-based implementation of this approach, by developing a preserving aspect-oriented .NET environment. This forwarded conceptualization to composite services with dynamic adaptability integrates thus most advanced software-engineering methods to evolving software, namely business rules, aspect-oriented concepts, architectural mechanisms and reflection techniques. Indeed, most existing proposals to software evolution including software as service, capitalize mostly on the integration of two mechanisms.

## Chapter 6

# Conclusions and Future Work

The main objective we have been focussing during this work concerned the disciplined modeling and dynamic adaptability, while developing knowledge-intensive service-oriented applications. We put forwards an integrated and progressive approach based on advanced software-engineering concepts including: Event-driven business rules and stereotyped UML constructs, aspect-oriented mechanisms, High-level Petri nets, architectural techniques and algebraic and rewriting techniques and logic. In this closing chapter, we first summarize the main achieved contributions. Secondly, we present some of possible further explorations towards a more complete approach and methodology.

### 6.1 Main achieved contribution

As the service-oriented paradigm with its Web-services technology are getting increasingly maturing, more and more (world-wide) cross-organizations and governmental institutions are shifting towards this technology at fast paces. This growing embracing of the service technology have been pressing for more disciplined engineering of complex and evolving composite service applications. First, most of potentials service applications such as E-Commerce, E-health and E-Government are by essence *knowledge-intensive*, mainly governed by multi-concern and agile event-driven business rules. Second, developed service-oriented applications are mostly mission critical, which implies precise and high degree of *formality* are to be ensured. Third, to cope with harsh competition and market volatility, modern service-oriented applications are imperatively dynamic and *adaptive*.

In this work, we have thus been addressing these three crucial challenging features, while developing today's complex service-oriented applications, that is, business rule-centricity, formal foundation and runtime adaptability. More precisely, we put forwards an integrated and progressive approach for specifying and validating service-oriented rule-centric and adaptive business applications. The approach is based on an innovative variant of high-level Petri nets, we referred to as adaptive CSRV-NETS. This framework can be distinguished at least with the following characteristics.



The fact that CSRV-NETS intrinsically permit 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 using any partial-ordering of transitions (as business-activities), that is, parallelism, sequence, choices and so on.

CSRV-NETS transitions are incrementally built to directly reflect corresponding event-driven ECA-driven business rules. In particular, any conditions what complex soever are straightforwardly constructed using first-order expressions on involved message parameters and service states.

As we demonstrated any CSRV-NETS web services specification can be concurrently validated, using their inherent semantics in terms of rewriting logic and tailored extension of the MAUDE language.

The forwarded approach promotes a two-level methodology for specifying and validated rule-centric service-oriented applications. That is, first, usual service orchestration is achieved by focussing on a specific service, while communicating with others. The second system-level allows specifying collaborating services as a choreography.

The most significant challenging contribution of this work concerns the dynamic adaptability. We thus put forward on top of each CSRV-NETS specification an extra aspectual-level and linked it to the base-level, in way that business rules can be dynamically shifted up and down.

In a summary the most significant contributions of this work with respect to the formal engineering of adaptive and knowledge-intensive service-oriented applications could be again highlighted in the following points:

**Incremental formalization and validation service-oriented applications** : The formalism is progressively constructed by first semi-formally modelling the concerned service-driven applications using stereo-typed UML class-diagrams and ECA-driven business rules. The CSRV-NETS formalism, as we just emphasized permits to intrinsically integrate such business rules in the modeled service-oriented business process. Furthermore, CSRV-NETS deal with both orchestration and choreography in a harmonious complementary manner.

**Inherent design-time rule-centric adaptability** : By inherently integrating event-driven business rules, the resulting CSRV-NETS specification is by construction very flexible and adaptable. Moreover, at design-time any modelled business rules-as-transition can be updated when requested. More precisely, we can change the conditions and /or any input / output arc-inscriptions of the chosen transition.

**Emerging runtime behavioral adaptability** : As we emphasized this is the most challenging, since no so far proposals have been able to cope with runtime service adaptability. Capitalizing on aspect-oriented mechanisms, we presented how rigorously and consistently weaving / unweaving any business rule at runtime.

**Compliant .NET Environment** : Besides the formal specification, validation and adaptability, we further experimented the deployment phase. For that purpose, we developed a compliant .NET environment that capitalize of aspect-oriented programming mechanisms to achieve the conceptualized dynamic adaptability.

## 6.2 Envisioned further investigations

After having put forwards 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.

**More case studies.** Along all main chapters, we have demonstrated the practicability of the proposed approach using a medium-size variant of the travel agency. Nevertheless, we are conscious that, for further enhancing the practicability and discovering domain-based patterns and specificities for the approach, more case studies covering different domains such E-health and E-government are necessary in any future step. The undertaking of others case-studies further allow for consolidating the approach and a general-purpose framework and discover any peculiarities that should require more perfection.

**Deployment using advanced Web standards.** In this work we have been concentrating on the foundational-level, though we developed a compliant .NET environment. Since this ultimate phase require several different experimentations, we are aware that further investigations are needed, by particularly using advanced web services technology and its standards. In this sense, we project that such deployment must addressed in more detail in any next extension of the proposed approach. In particular, one a specification is corrected, validated and adapted it should be mapped to corresponding tailored web standards such as and BPEL and WS-CDL. 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 [RD05] could a promising starting point. Another interesting direction towards that aim is the adoption of aspect-oriented techniques as recently suggested in [CKM07].

**Supporting tools for the approach.** As next promising towards enhancing the practicability of this approach belongs the development of supporting software tools. These tools should include at least an editor-simulator for CSRV-NETS specification, that is, this tool should allow the designer to describe his specification, correct it and validate it using graphical simulation as we highlighted in the work. The second complementing tool should cope with the runtime adaptability, that is, it should permits the manipulation of business rules as tuples, dynamically bringing them down to the

base-level and animating them as we gave in the previous chapter. The respective formal MAUDE aspect-orientation require to be integrated with such envisioned CSRV-NETS tools, so that the formal validation and reasoning could directly automated. Last but not least, we aim investigating the tailored of service standards using the development .Net environment.

**Extensions towards formal verification.** It would be promising direction to extend this work towards verification, instead of just modelling and validation. More specification, we argue that the research we initiated in [AS08b] in leveraging Lamport's TLA logic [Lam94], could further investigated towards composing and verifying behavioral Web-services. Indeed, we claim that it is quite possible since CSRV-NETS transitions are governed by rewriting rules, we could refine as TLA formulas in that recent work.

**Multi-dimensional rule-based service development.** We already hinted that the adopted ECA-driven rules could be specialized for different concerns such as: Context-awareness, management and qualities and / or security and privacy. Nevertheless, we did not developed further on such rule-centric separation of concerns. For instance, in [ABS09a, Aou09] we detail at the descriptive-level, how context-awareness concerns as tailored ECA-driven rules, can be explicitly separated from usual functionality concerns. A promising direction we are thus intensively exploring consists in leveraging such descriptive results at the foundation and adaptive levels, following the forwarded approach. In such manner, every concerns become adaptive on its own-level and during the integration of concerns.

# Bibliography

- [Aa02a] A. Ankolekar and et al. DAML-S:Web Service Description for the Semantic Web. In *Proc. of International Semantic Web Conference (ISWC)*, pages 348–363. IEEE CS, 2002.
- [AA02b] G. Antoniou and M. Arief. Executable declarative business rules and their use in electronic commerce. In *Proceedings of the 2002 ACM symposium on Applied computing (SAC '02)*, pages 6–10. ACM Press, 2002.
- [Aal03] W.M.P. Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18:72–76, 2003.
- [AB04] S. Askary and B. Bloch. Web service business process execution language version 2.0. Technical report, OASIS, <http://www.oasis-open.org/apps/org/workgroup/wsbpel/>, December 2004.
- [ABFL07] J. Abreu, L. Bocchi, J.L. Fiadeiro, and A. Lopes. Specifying and Composing Interaction Protocols for Service-Oriented System Modelling. In *Formal Techniques for Networked and Distributed Systems - FORTE 2007*, volume 4574 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2007.
- [ABS00] N. Aoumeur, K. Barkaoui, and G. Saake. On the Benefits of Rewrite Logic as a Semantics for Algebraic Petri Nets in Computing Siphons and Traps. In *Proc. of the 10th International Conference on Computing and Information (ICCI '2000)*, Kuwait, 2000. to appear in LNCS, Springer.
- [ABS09a] N. Aoumeur, K. Barkaoui, and G. Saake. A Multi-Dimensional Architectural Approach to Behavior-Intensive Adaptive Pervasive Applications. In *Proc. of 4th International Symposium on Wireless Pervasive Computing (ISWPC'09)*, pages 1–8. IEEE CS Press, 2009.
- [ABS09b] N. Aoumeur, K. Barkaoui, and G. Saake. Rapid-prototyping of Adaptive Component-based Systems using Runtime Aspectual Interactions. In *Proc. of 20th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP'09)*, pages 18–25. IEEE CS Press, 2009.
- [ABS09c] N. Aoumeur, K. Barkaoui, and G. Saake. Towards a Disciplined Engineering of Adaptive Service-oriented Business Processes. In *Proc. of 4th IEEE International Conference on Internet and Web Applications and Services (ICIW'09)*, pages 474–480. IEEE CS Press, 2009.
- [ACKM04] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer-Verlag, 2004.
- [AFO06] N. Aoumeur, J. Fiadeiro, and C. Oliveira. Distribution Concerns in Service-Oriented Modelling. *International Journal of Internet Protocol Technology (IJPT)*, 1(3):144–158, 2006.
- [Aou02] N. Aoumeur. *Specifying and Validating Consistent and Dynamically Evolving Concurrent Information Systems: An Object Petri-net Based Approach*. Shaker-Verlag, 2002. ISBN 3-8265-9971-3.
- [Aou09] N. Aoumeur. Architectural Specification of Context-aware Service Applications in Rewriting logic. Technical Report, ITI, FIN, Universitaet Magdeburg, 2009. *Submitted for Publication*.
- [Ars04] A. Arsanjani. Service-Oriented Modeling and Architecture (SOMA). Report, <http://www.ibm.com/developerworks/webservices/library/ws-soa-design>, IBM Developer-Works, 2004.

- [AS02] N. Aoumeur and G. Saake. A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems. *Data and Knowledge Engineering*, 42(2):143–187, August 2002.
- [AS04] N. Aoumeur and G. Saake. Dynamically Evolving Concurrent Information Systems: A Component-Based Petri Net Proposal. *Data and Knowledge Engineering*, 50(2):117–173, 2004.
- [AS08] N. Aoumeur and G. Saake. Modelling and Certifying Concurrent Systems: a MAUDE-TLA Driven Architectural Approach. In *In Proc. of 5th International Conference on Information Technology : New Generations (ITNG'08)*. IEEE CS, 2008.
- [ASB07] N. Aoumeur, G. Saake, and K. Barkaoui. Incremental Specification Validation and Runtime Adaptivity of Distributed Component Information systems. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 123–136. IEEE Computer Society, 2007.
- [AW02] T. Archer and A. Whitechapel. *Inside Microsoft C#*. Microsoft Press, 2002.
- [BB91] M. Baldassari and G. Bruno. PROTOB : an Object Oriented Methodology for Developing Discrete Event Dynamic Systems. *Computing Languages*, 16:39–63, January 1991.
- [BBBea05] M. Beisiegel, H. Blohm, D. Booz, and et al. Service Component Architecture. Building Systems using a Service Oriented Architecture. Technical report, [ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA-White-Paper1-09.pdf](http://ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA-White-Paper1-09.pdf), 2005.
- [BBG97] O. Biberstein, D. Buchs, and N. Guelfi. CO-OPN/2: A Concurrent Object-Oriented Formalism. In *Proc. of Second IFIP Conf. on Formal Methods for Open Object-Based Distributed Systems(FMOODS)*, pages 57–72. Chapman and Hall, March 1997.
- [BCG+06] T. Batista, C. Chavez, A. Garcia, A. Sant'Anna, U. Kulesza, A. Rashid, and F. Filho. Reflections on Architectural Connection: Seven Issues on Aspects and ADLs. In *Proc. of Workshop on Early Aspects at ICSE'06*, 2006.
- [BDO05] A. Barros, M. Dumas, and P. Oaks. Standards for Web Service Choreography and Orchestration: Status and Perspectives. In *Proc. of Business Process Management (BPM 2005)*, pages 61–74. LNCS, Vol. 3812, 2005.
- [BGCM94] E. Battiston, V. Grespi, F. D. Cindio, and G. Mauri. Semantics Frameworks for a Class of Modular Algebraic Nets. In M. Nivat, C. Rattray, T. Russ, G. Scollo, editor, *Proc. of 3th. International AMAST Conference*, 1994.
- [BGG+05] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and Orchestration: a synergic approach for system design. In *Proc. of 3rd International Conference on Service Oriented Computing (ICSOC'05)*, pages 228–240. LNCS vol. 3826. Springer, 2005.
- [BJR98] G. Booch, I. Jacobson, and J. Rumbaugh, editors. *Unified Modeling Language, Notation Guide, Version 1.0*. Addison-Wesley, 1998.
- [BK05] M. Bajec and M. Krisper. A Methodology and Tool Support for Managing Business Rules in Organisations. *Information Systems*, 30(6):423–443, 2005.
- [BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
- [BMSB93] M. Bettaz, M. Maouche, M. Soualmi, and S. Boukebeche. Protocol Specification using ECATNets. *Reséaux et Informatique Répartie*, 3(1):7–35, 1993.
- [Boo04a] D. Booth. Web Services Architecture. *W3C Working Group Note*, <http://www.w3.org/TR/ws-arch/>, 2004.
- [Boo04b] David Booth. Web Services Architecture W3C Working Group Note, February 2004. <http://www.w3.org/TR/ws-arch/>.
- [Bru02] Robert Brunner. *Web services and Flows (WSFL)*. Sams Publishing, [www.developer.com/java/web/article.php/1462301](http://www.developer.com/java/web/article.php/1462301), September 2002.

- [CCL00] W. Cazzola, S. Chiba, and T. Ledoux. Reflection and meta-level architectures: State of the art, and future trends. In J. Malenfant, S. Moisan, and A. Moreira, editors, *ECOOP'2000 Workshop Reader*, volume 1964 of *lncs*, pages 1–15. Springer, 2000.
- [CDE<sup>+</sup>07] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and C.L: Talcott. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. *Lecture Notes in Computer Science (springer)*, 4350, 2007.
- [CF05] C. Courbis and A. Finkelstein. Towards aspect weaving applications. In *Proceedings of the 27th international conference on Software engineering (ICSE '05)*, pages 69–77. ACM Press, 2005.
- [CG01] S. Cheng and D. Garlan. Mapping Architectural Concepts to UML-RT. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '2001)*, 2001.
- [CGK<sup>+</sup>04] F. Curbera, Y. Golland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Languagefor Web Services (BPEL4WS 1.1). <http://www-106.ibm.com/webservices/ws-bpel>, IBM report, 2004.
- [CGS01] F. Casati, D. Georgakopoulos, and M.C. Shan. Technologies for E-Services. In *Proc. of 2nd International Workshop on Technologies for E-Services*, pages 16–29. LNCS, Vol. 2193, 2001.
- [CGS02] W. Cazzola, A. Ghoneim, and G. Saake. Reflective Analysis and Design for Adapting Object Run-time Behavior. In Zohra Bellahsene, Dilip Patel, and Colette Rolland, editors, *Proceedings of the 8th International Conference on Object-Oriented Information Systems (OOIS'02)*, Lecture Notes in Computer Science 2425, pages 242–254. Springer-Verlag, on 2nd-5th of September 2002. ISBN: 3-540-44087-9.
- [CKM07] A. Charfi, R. Khalaf, and N. Mukhi. QoS-Aware Web Service Compositions Using Non-intrusive Policy Attachment to BPEL. In *In Proc. of International Conference on Service Oriented Computing (ICSOC'07)*, volume 4749 of *Lecture Notes in Computer Science*, page 582593. Springer, 2007.
- [CM96] M. Clavel and J. Meseguer. Reflection and Strategies in rewriting logic. In G. Kiczales, editor, *Proc. of Reflection'96*, pages 263–288. Xerox PARC, 1996.
- [CM04] A. Charfi and M. Mezini. Hybrid web service composition: Business processes meet business rules. In *Proceedings 2nd International Conference on Service Oriented Computing (ICSOC04)*. ACM Press, 2004.
- [CM07] A. Charfi and M. Mezini. AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web Journal: Recent Advances on Web Services (special issue)*, 10:309–344, 2007.
- [DD04] R. Dijkman and M. Dumas. Service-Oriented Design: A Multi-Viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
- [DFS04] R. Douence, P. Fradet, and M. Sudholt. Composition, reuse and interaction analysis of stateful aspects. In *In Proc. In 4th Int. Conf. on Aspect-oriented Software Development (AOSD'04)*, pages 141–150. ACM Press, 2004.
- [DGS04] Skogan D., R. Grønmo, and I. Solheim. Web Service Composition in UML. In *8th International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 47–57. IEEE Computer Society, 2004.
- [DKL<sup>+</sup>08] G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and Weske. M. Modeling Service Choreographies Using BPMN and BPEL4Chor. In *Proc. of Advanced Information Systems Engineering, 20th International Conference (CAiSE'08)*, pages 79–93. LNCS, Vol. 5074, Springer, 2008.
- [DW07] Kopp O. Leymann F. Decker, G. and M. Weske. BPEL4Chor: Extending BPEL for modeling choreographies. In *Proc. International Conference on Web Services*, pages 296–303. IEEE, 2007.

- [EFB01] T. Elrad, R. Filmanand, and A. Bader. (guest editors). *Communications of the ACM (Special Issue on Aspect Oriented Programming)*, 44(10), 2001.
- [EM85] H. Ehrig and B. Mahr. Fundamentals of algebraic specifications 1 : Equations and initial semantics. *EATCS Monographs on Theoretical Computer Science*, 21, 1985.
- [EM06] A. Erradi and V. Maheshwari, P. Tosic. Policy-Driven Middleware for Self-adaptation of Web Services Compositions. In *ACM/IFIP/USENIX 7th International Middleware Conference*, volume 4290 of *Lecture Notes in Computer Science*, pages 62–80. Springer, 2006.
- [ETM07] A. Erradi, V. Tosic, and P. Maheshwari. MASC -.NET-Based Middleware for Adaptive Composite Web Services. In *IEEE International Conference on Web Services (ICWS'07)*, pages 727–734. IEEE Computer Society, 2007.
- [FECA04] R. Filman, T. Elrad, S. Clarke, and M. Aksit. *Aspect-Oriented Software Development*. Reading, MA: Addison Wesley, 2004.
- [FGV04] R. Farahbod, U. Glaesser, and M. Vajihollahi. Specification and Validation of the Business Process Execution Language for Web Services. In W. Zimmermann and B. Thalheim, editors, *Proc. of ASM'2004*, volume 3052 of *Lecture Notes in Computer Science*, pages 78–94. Springer, 2004.
- [FS07] L. Fuentes and P. Sánchez. Towards executable aspect-oriented UML models. In *Proceedings of the 10th international workshop on Aspect-oriented modeling (AOM'07)*, pages 28–34. ACM Press, 2007.
- [GCB+06] A. Garcia, C. Chavez, V. Batista, C. SantAnna, U. Kulesza, R. Awais, and C. Pereira de Lucena. On the Modular Representation of Architectural Aspects. In *Third European Workshop On Software Architecture, EWSA 2006*, volume 4344 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2006.
- [GLC99] B.M. Grosf, Y. Labrou, and H.Y. Chan. A declarative approach to business rules in contracts: courteous logic programs in XML. In *Proceedings of the 1st ACM conference on Electronic commerce (EC'99)*, pages 68–77. ACM Press, 1999.
- [Gri98] F. Griffel. *Componentware : Konzepte und Techniken eines Softwareparadigmas*. Dpunkt-Verlag, Heidelberg, 1998.
- [GS03] G.Meredith and S.Bjorg. Contracts and Types. In M.Papazoglou and D.Georgakopoulos (guest editors), editors, *Special Issue on Service-Oriented Computing, Communications of the ACM*, volume 46(10), pages 41–47. 2003.
- [HB03] R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. In K. Schewe and X. Zhou, editors, *Proceedings of the 14th Australasian Database Conference*, volume 17 of *CRPIT*, pages 191–200. Australian Computer Society, 2003.
- [HBM08] R. Hamadi, B. Benatallah, and B. Medjahed. Self-adapting recovery nets for policy-driven exception handling in business processes. *Distributed Parallel Databases*, 23:1–44, 2008.
- [HHJT98] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about Classes in Object-Oriented Languages: Logical Models and Tools. In Ch. Hankin, editor, *European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 1998.
- [HHL05] J.H. Hausmann, R. Heckel, and M. Lohmann. Model-based development of web service descriptions: Enabling a precise matching concept. *International Journal of Web Services Research*, 2(2):67–84, 2005.
- [HJL+07] J. Han, Y. Jin, Z. Li, T. Phan, and J. Yu. Guiding the Service Composition Process with Temporal Business Rules. In *Proc. of International Conference on Web Services (ICWS'07)*, pages 735–742. IEEE CS Press, 2007.
- [Hun04] D Hunter. *Beginning XML*. Wiley Publishing, 2004.
- [Jen92] K. Jensen. Coloured Petri Nets: Basic Concepts, Analysis Methods and practical Use - Volume 1 : Basic Concepts. *EATCS Monographs in Computer Science*, 26, 1992.

- [JFT07] I.J. Jureta, S. Faulkner, and P. Thiran. Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems. In *In Proc. of 5nd International Conference on Service Oriented Computing (ICSOC'07)*, volume 4749 of *Lecture Notes in Computer Science*, pages 270–282. Springer, 2007.
- [JPHL07] J.Y. Jung, J. Park, S. Han, and L. Lee. An ECA-based framework for decentralized coordination of ubiquitous web services. *Information Software Technology*, 49(11-12):1141–1161, 2007.
- [JR91] K. Jensen and G. Rozenberg. *High-level Petri Nets*. Springer, 1991.
- [Kea97] G. Kiczales et al. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. LNCS, 1997.
- [Kea01] G. Kiczales et al. An Overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'01)*, pages 327–353. LNCS 2072, 2001.
- [KEP00] G. Knolmayer, R. Endl, and M. Pfahner. Modeling Processes and Workflows by Business Rules. In W. van der Aalst et al., editor, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2000.
- [KL04] P. Kardasis and P. Loucopoulos. Expressing and Organising Business Rules. *Information and Software Technology*, 2004.
- [KOMC04] N. Kavantzias, G. Olsson, J. Mischkinsky, and M. Chapman. Web Services Choreography Description Language (WS-CDL) 1.0. <http://www.w3.org/tr/ws-cdl-10>, w3.org, 2004.
- [Kre01] Heather Kreger. Web services conceptual architecture ( wsca 1.0). Technical report, IBM Software Group, <http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, May 2001.
- [Lak96] C. Lakos. 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.
- [Lam94] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
- [LLGL08] H. Liu, Q. Li, N. Gu, and A. Liu. Exploiting Semantics for Analyzing and Verifying Business Rules in Web Services Composition and Contracting. In *Proc. of International Conference on Web Services (ICWS'08)*, pages 112–119. IEEE CS Press, 2008.
- [LMSW08] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data & Knowledge Engineering*, 64, 2008.
- [LOS97] P. Lang, W. Obermair, and M. Schrefl. 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.
- [Mar03] A. Martens. On Usability of Web Services. In Calero and Diaz and Piattini, editor, *Proc. of 1st Web Services Quality Workshop (WQW 2003)*, 2003.
- [Mar06] D. Martin. Putting Web Services in Context. *Electronic Notes in Theoretical Computer Science*, 146:3–16, 2006.
- [MBM<sup>+</sup>07] H. Motahari, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-Automated Adaptation of Service Interactions. In *The 16th International World Wide Web Conference (WWW2007)*, 2007.
- [MDM09] S. McIlvenna, M. Dumas, and Wynn. M.T. Synthesis of Orchestrators from Service Choreographies. In *Conceptual Modelling 2009, Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009)*, pages 129–138. Australian Computer Society, Vol. 96, CRPIT, 2009.
- [Mes92] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. *Theoretical Computer Science*, 96:73–155, 1992.



- [MOM96] N. Marti-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In J. Meseguer, editor, *Proc. of First International Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 189–224, 1996.
- [MOO04] D. Moldt, S. Offermann, and J. Ortmann. A Proposal for Petri Net Based Web Service Application Modeling. In *In Proceedings of ICWE 2004*, volume 4140 of *Lecture Notes in Computer Science*, pages 93–97. Springer, 2004.
- [MOPF<sup>+</sup>05] N. Marti-Oliet, I. Pita, J.L. Fiadeiro, J. Meseguer, and T.S.E. Maibaum. A Verification Logic for Rewriting Logic. *J. Log. Comput.*, 15(3):317–352, 2005.
- [MPP02] M. Mecella, F.P. Presicce, and B. Pernici. Modeling E-service Orchestration through Petri Nets. In A. Buchmann et al., editor, *TES 2002*, volume 2444, pages 38–47, 2002.
- [MRS05] P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.
- [MS04] K. Mahbub and G. Spanoudakis. A Framework for Requirements Monitoring of Service Based Systems. In M. Aioello, M. Aoyama, F. Curbera, and M. Papazoglou, editors, *2nd International Conference on Service Oriented Computing (ICSOC'04)*. ACM Press, 2004.
- [MSK08] P. Mayer, A. Schroeder, and N. Koch. A Model-Driven Approach to Service Orchestration. In *Proceedings of the IEEE International Conference on Services Computing (SCC'08)*. IEEE Computer Society, 2008.
- [MT00] N. Medvidovic and R.N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.
- [New02] Eric Newcomer. *understanding web services*. Addison-Wesley, September 2002.
- [NM03] S. Narayanan and S. McIlraith. Analysis and Simulation of Web Services. *Computer Networks*, 42:675–693, 2003.
- [NRD06] C. Nagl, F. Rosenberg, and S. Dustdar. ViDRE - A Distributed Service-Oriented Business Rule Engine based on RuleML. In *10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*. IEEE Computer Society, 2006.
- [OH06] G. Ortiz and J. Hernandez. Toward UML Profiles for Web Services and their Extra-Functional Properties. In *IEEE International Conference on Web Services (ICWS'06)*, pages 889–892. IEEE Computer Society, 2006.
- [OMG01] OMG. *The Common Object Request Broker : Architecture*. Object Management Group, 2.4 Edition, Nov.2000, <http://www.omg.org/technology/documents/formal/corbaaiop.htm>, 2001.
- [OMG05] OMG. UML 2.0: Superstructure Specification. Version 2.0, formal/05-07-04. Technical report, omg.org, 2005.
- [OVvdA<sup>+</sup>07a] C. Ouyang, E. Verbeek, V.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Sciences of Computer Programming*, 67(2-3):162–198, 2007.
- [OVvdA<sup>+</sup>07b] C. Ouyanga, E. Verbeekb, V.M.P. van der Aalst, F. Breutela, M. Dumas, and A.H.M. Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Science of Computer Programming*, 67:162–198, 2007.
- [owl04] *Bringing Semantics to Web Services: The OWL-S Approach*, volume 3387 of *LNCS*. Springer, 2004.
- [OYP03] B. Orriëns, J. Yang, and M.P. Papazoglou. A Framework for Business Rule Driven Web Service Composition. In *Proc. of Conceptual Modeling for Novel Application Domains*, volume 2814 of *Lecture Notes in Computer Science*, pages 52–64. Springer, 2003.
- [Pap07] M.P. Papazoglou. *Web Service: Principles and Technology*. Prentice-Hall, Englewood Cliffs, 2007.

- [PD99] N.W. Paton and O. Dyaz. Active Database Systems. *ACM Computing Surveys*, 31(1):63–103, 1999.
- [Pel03] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.
- [PPSGS04] V. Poladian, J. Pedro Sousa, D. Garlan, and M. Shaw. Dynamic Configuration of Resource-Aware Services. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*. ACM Press, 2004.
- [PTDL07] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.
- [PTDL08] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *Int. J. Cooperative Inf. Syst.*, 17(2):223–255, 2008.
- [RD05] F. Rosenberg and S Dustdar. Towards a Distributed Service-Oriented Business Rules System. In *Proc. of the of EEE European Conference on Web services (ECOWS)*. IEEE Computer Society Press, 2005.
- [Rei85] W. Reisig. Petri Nets. *EATCS Monographs on Theoretical Computer Science*, 4, 1985.
- [Rei91] W. Reisig. Petri Nets and Abstract Data Types. *Theoretical Computer Science*, 80:1–30, 1991.
- [RIM08] I. Rauf, M.Z.Z. Iqbal, and Z.I. Malik. UML Based Modeling of Web Service Composition - A Survey. In *Proc. of Software Engineering Research, Management and Applications (SERA'08)*, pages 301–307. IEEE Computer Society, 2008.
- [RLA<sup>+</sup>09] S. Rahman, A. Lodhi, N. Aoumeur, C. Rautenstrauch, and G. Saake. Intra-Service Adaptability for ECA-Centric Web Services using Contract and Aspect. In *Proc. of IADIS International Conference Information Systems 2009 (IS'09)*, pages 135–142. IADIS Press, 2009.
- [Rul05] RuleML. The Rule Markup Initiative. *www.ruleml.org*, 2005.
- [RW02] D. Rosca and C. Wild. Towards a Flexible Deployment of Business Rules. *Expert Systems with Applications*, 23:385–394, 2002.
- [SB94] C. Sibertin-Blanc. Communicative and cooperative nets. In E. Astesiano, R. Reggio, and A. Tarlecki, editors, *Proc. of the 15th International Confernce on the application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*. Springer, 1994.
- [SB05] Q.Z. Sheng and B. Benatallah. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development. In *International Conference on Mobile Business (ICMB'05)*, pages 206–212. IEEE CS Press, 2005.
- [SCZ04] M. Solanki, A. Cau, and H. Zedan. Introducing Compositionality in Web Service Descriptions. In *Proceedings of the International Conference on World Wide Web*. IEEE Computer Society Press, 2004.
- [SG96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
- [SGS04] D. Skogan, R. Groenmo, and R. Solheim. Web service composition in uml. *IEEE international*, pages 47–57, Sept 2004.
- [SS99] J.G. Schmolzea and W. Snyder. Detecting redundancy among production rules using term rewrite semantics. *Knowledge-Based Systems*, 12(1-2):3–11, 1999.
- [ST01] Andrew. S. and M.V.S Tanenbaum. *Distributed Systems:Principles and Paradigms*. Prentice Hall PTR, 2001.
- [Ste98] M.-O. Stehr. A Rewriting Semantics for Algebraic Petri Nets. Manuscript, SRI International, March 1998.
- [TBL01] Ora Lassila T. Berners-Lee, J.Hendler. *The Semantic Web*. Scientific Amrican, May 2001.

- [Tec04] Technical. Uddi technical white paper. Technical report, OSSIS, <http://uddi.org/pubs/uddi-tech-wp.pdf>, October 2004.
- [Tha01] S. Thatte. Xlang web services for business process design, 2001.
- [TWB03] S. Tabet, G. Wagner, and H. Boley. MOF-RuleUML: The Abstract Syntax of RuleML as a MOF Model. In N. Lenehan, editor, *Integrate 2003*, 2003.
- [URAS09] S. Ur Rahman, N. Aoumeur, and G. Saake. An Adaptive ECA-Centric Architecture for Agile Service-Based Business Processes with Compliant Aspectual .NET Environment. In *Proc. of ACM The 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008)*, pages 240–247. ACM Press, 2009.
- [Val01] Ruediger Valk. Concurrency in communicating object petri nets. In G. Agha, F. de Cindio, and G. Rozenberg, editors, *Concurrent Object Oriented Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*, pages 164–165. Springer, 2001.
- [VCJ03] B. Verheecke, M.A. Cibran, and V. Jonckers. AOP for Dynamic Configuration and Management of Web Services. In *In Proc. of International Conference on Web-Services (ICWS-Europe'03)*, pages 137–151. LNCS, Volume 2853, 2003.
- [VdABHK00] W.M.P. Van der Aslst, A. Barros, H.M. Hofstede, and B. Kiepuszewski. Advanced workflow patterns. In *Proc. of COOPIS'2000*, pages 18–29, 2000.
- [vdAWMO<sup>+</sup>06] van der Aalst. W., Dumas. M., C. Ouyang, A. Rozinat, and H.M.W. Verbeek. Choreography Conformance Checking: An Approach based on BPEL and Petri Nets. In *The Role of Business Processes in Service Oriented Architectures*. Dagstuhl Seminar Proceedings, 2006.
- [Ved01] A.S. Vedamuthu. *Web Services Description Language (1.1)*. W3C Recommendation, <http://www.w3.org/TR/wsdl>, March 2001.
- [W3C04] W3C. Web Services Choreography Description Language Version 1.0. *IEEE Computer*, <http://www.w3.org/TR/2004/WD-wscdl-10-20041217>, 2004.
- [WCL<sup>+</sup>05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and DF Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WSReliable Messaging, and More*. Prentice-Hall, Englewood cliffs, 2005.
- [Web08] Web. Web-Services Architect, Part 2: Models for dynamic-business. *Web*, <http://www-106.ibm.com/developerworks/webservices/library/ws-arc2.html>, 2008.
- [Weg90] P. Wegner. Concepts and paradigms of Object-Oriented Programming. *OOPS Messenger*, 1:7–87, 1990.
- [WFF] WFF:. Windows Workflow Foundation. Technical report, <http://netfx3.com/content/WFHome.aspx>.
- [WKL03] W.M.N. Wan-Kadir and P. Loucopoulos. Relating Evolving Business Rules to Software Design. *Journal of Systems Architecture*, 2003.
- [WSHG06] Y. Wang, S. Singh, J. Hosking, and J. Grundy. An aspect-oriented UML tool for software development with early aspects. In *Proceedings of the 2006 international workshop on Early aspects at ICSE*, pages 51–58. ACM Press, 2006.
- [YK04] X. Yi and K.J. Kochut. A CP-nets-based Design and Verification Framework for Web Services Composition. In *In Proceedings of 2004 IEEE International Conference on Web Services*, pages 756–760. IEEE Computer Society, 2004.
- [YK05] X. Yi and K.J. Kochut. A CPNets-based Framework for Design and Analysis for Service Oriented Distributed Systems. *ACM Transaction on Internet Technology*, ??, 2005.
- [YM01] A. Yonezawa and S. Matsuoka, editors. *Metalevel Architectures and Separation of Crosscutting Concerns*. Springer, Volume 2192, Proc. REFLECTION 2001, 2001.
- [YTX05] Y. Yang, Q. Tan, and Y. Xiao. Verifying Web Services Composition Based on Hierarchical Colored Petri Nets. In *Proceedings of IHIS05*, pages 47–53. ACM Press, 2005.

## Appendix A

# Algebraic Specifications, (High-level) Petri Nets, Rewriting logic and MAUDE: Overview

As sketched in the previous chapter the approach we are proposing in this thesis is to leverage rewriting techniques to aspectual architectural level. Besides that, this approach is mainly conceived for specifying and validating advanced information system. In this chapter we will first recall some concepts from algebraic abstract data type and rewriting techniques, and then rewriting logic is introduced. In the rewriting logic, concurrent computation by rewriting coincides with logical deduction.

### A.1 Algebraic specification: an overview

Abstract data type are ubiquitous in different programming and specification paradigms. They allow describing a class of data domains. It is therefore desirable to find a way of characterizing 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 [EM85, ?].

**Definition A.1.1 (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  $(f : s_1 \times \dots \times s_n \rightarrow s_0) \in F$  for the same symbol  $f$  and different arities  $(s_1 \times \dots \times s_n \rightarrow s_0)$  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 A.1.2** The following algebraic signature  $Nat$  defines the syntax of natural numbers.

```
obj NAT is
  sorts Nat .
  op 0 : → Nat .
  op succ : Nat → NzNat .
```

**op** + :  $\text{Nat} \times \text{Nat} \rightarrow \text{Nat}$  .  
**endo**

**Definition A.1.3 (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  $\{\mathcal{A}_s | s \in S\}$  of subsets of  $\mathcal{A}$  such that  $\mathcal{A} = \bigcup_{s \in S} \mathcal{A}_s$ , with an interpretation of each operator  $(f : s_1 \times \dots \times s_n, s) \in F$  as a partial function  $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \dots \times \mathcal{A}_{s_n} \rightarrow \mathcal{A}_s$  such that:

- $\mathcal{A}_s \in \mathcal{A}_{s'}$  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  $f^{\mathcal{A}} : \vec{\mathcal{A}}_{\vec{s}'_1} \rightarrow \mathcal{A}_{s'_0}$
- the class of all *Sig*-algebras is denoted by  $\text{Alg}(\text{Sig})$ .

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  $\text{sort}(x)$ .  $\square$

**Definition A.1.4 (Terms and Term-Algebra):** Let  $\text{Sig} = (S, \leq, F)$  be a signature.

- (1) A term of sort  $s \in S$  is either a variable  $x$  with  $\text{sort}(x) \leq s$  or it has the form  $f(t_1, \dots, t_n)$ , where  $(f : s_1 \times \dots \times s_n \rightarrow s_0) \in F$  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 *simple a term*).
- (3) The set of all open terms over *Sig* with variables from a variable set  $X$  is denoted  $T_{\text{Sig}}(X)$ . The set of open terms over *Sig* of sort  $s$  with variables from  $X$  is denoted  $T_{\text{Sig}}^S(X)$ . The set  $T_{\text{Sig}}(\phi)$  of all closed terms is denoted  $T_{\text{Sig}}$ , and the closed terms of sort  $s$  are  $T_{\text{Sig}}^S$ .
- (4)  $T_{\text{Sig}}$  is a *Sig*-algebra that takes:  $(T_{\text{Sig}})_s := T_{\text{Sig}}^S, f^{T_{\text{Sig}}}(t_1, \dots, t_n) := f(t_1, \dots, t_n)$ .  $T_{\text{Sig}}$  is called the *term algebra of Sig*.

► **Example A.1.5** the term algebras of the natural number  $\text{Nat}$  for example:

$$T_{\text{Nat}} = \{0 ; \text{succ}(0) ; \text{succ}(\text{succ}(0)) ; \dots ; +(\text{succ}(\text{succ}(0)), 0) ; \text{succ}(+(\text{succ}(0), \text{succ}(\text{succ}(\text{succ}(0)))))) \dots \}$$

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 A.1.6 (Assignment and Substitution):** Let  $\mathcal{A}$  be a *Sig*-algebra and  $X$  be a set of variables.

- (1) An *assignment*  $\text{ass} : X \rightarrow \mathcal{A}$  is a function with  $\text{ass}(x) \in \mathcal{A}_{\text{sort}(x)}$ .
- (2) Let  $t$  be a *Sig*-term containing only variables from  $X$ . The *denotation*  $\text{ass}(t)^{\mathcal{A}}$  is defined by induction on the structure of  $t$ :
  - (a)  $\text{ass}(x)^{\mathcal{A}} = \text{ass}(x)$ ,
  - (b)  $\text{ass}(f(t_1, \dots, t_n))^{\mathcal{A}} = f^{\mathcal{A}}(\text{ass}(t_1)^{\mathcal{A}}, \dots, \text{ass}(t_n)^{\mathcal{A}})$ . if  $t$  is a ground term we write  $t^{\mathcal{A}}$  instead of  $\text{ass}(x)^{\mathcal{A}}$ .

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

**Definition A.1.7 (Equations):** Let  $Sig = (S, \leq, F)$  be a signature. An *equation* over  $Sig$  has the form  $(D, C, e)$ , where:

$D = \{x_1 : s_1, \dots, x_n : s_n\}$  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*.  $\square$

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  $D$ .

**Definition A.1.8 (Order-Sorted Specification):** An *order-sorted specification*  $Spec = (S, \leq, F, E)$  consists of a signature  $(S, \leq, F)$  and a set  $E$  of equations over this signature. By  $T_{Spec}(X)$  we refer to the term algebra over the signature of  $Spec$ .

► **Example A.1.9** The Nat signature can be extended to a Nat specification adding suitable equations.

```
obj NAT is
  sorts Nat .
  op 0 : → Nat .
  op succ : Nat → NzNat .
  op + : Nat × Nat → Nat .
  var n : Nat .
  var m : Nat .
  eq +(n, 0) = n .
  eq +(succ(n), m) = succ(+(n, m)) .
endo
```

The terms of sort Nat are generated by the constant  $0$  and the function  $succ$ , using data items.  $0$  and  $succ$  are called *generator* or *constructor* functions.  $+$  is a *projection* or *defined* function.

## A.2 (High-level) Petri-Nets: Main Concepts

In 1962, Petri Nets was first introduced by Carl Adam Petri in his Phd work "Kommunikation mit automaten"[?]. Here are many reasons that make Petri nets one of the leading framework for describing and analysing behavioral aspects in different kinds of concurrent and distributed systems. Indeed, Petri nets have been used to model and analyze complex applications in a wider variety of domains such as distributed software systems, business processes, telecommunication for designing and analysing protocols, information systems.

- 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 behavior of a Petri net ranging from sequential, interleaving, pomset 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 arcs) which is easy to comprehend and has therefore some wide appeal for practitioners.
- By their representation as directed, connected, and bipartite graphs, Petri nets have useful links both to graph theory and to linear algebra which can be exploited for the verification of systems (e.g. reachability, deadlock, and liveness).

### A.2.1 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, disjoint 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)  $W : F \rightarrow \mathbb{N}/0$ , attaches a weight to each arc of the net,
- (iv)  $M : S \rightarrow \mathbb{N}$ , 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 A.2.1 (the dining philosophers)** The left hand-side of Figure A.1 shows the well-known example of the five dining philosophers. In this net each philosopher  $P_i$  (with  $i \in \{1, \dots, 5\}$ ) may be in one of the two states, either eating or thinking, corresponding respectively to (presence of a token in) the places  $P_{i\_E}$  and  $P_{i\_T}$ . Each fork is modeled by a corresponding place, where the presence of a token indicates the availability of the fork. For each philosopher there are two actions (i.e. transitions): 'thinking or eating'. 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. ♦

### Definition A.2.2 (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
 
$$\forall s \in \bullet t : M(p) \geq W(p, t).$$

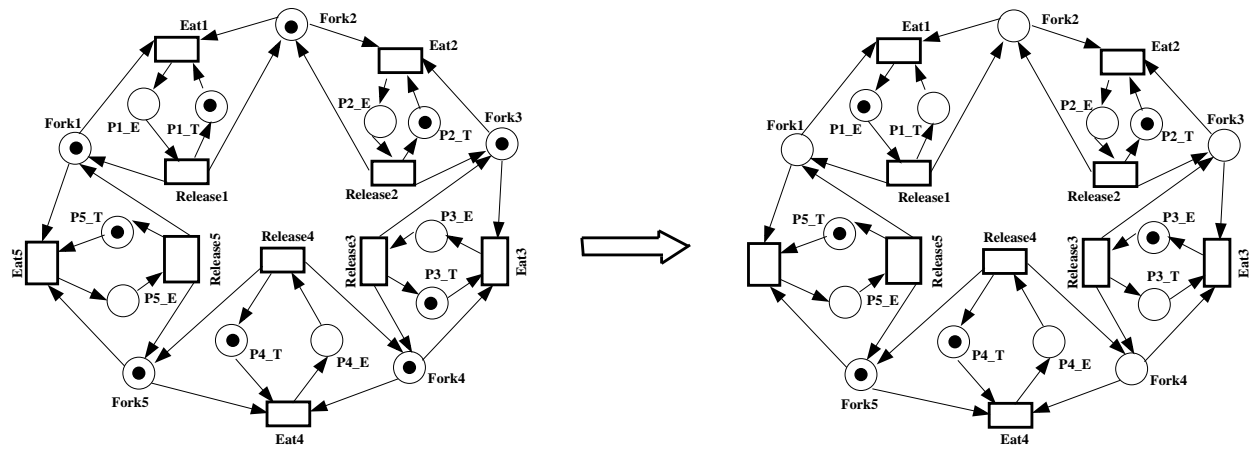


Figure A.1: The dining philosopher problem as a P / T-net.

- (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  $p \in P$ ,

$$M'(p) = \left\{ \begin{array}{ll} M(p) - W(p, t) & \text{iff } p \in \bullet t / t \bullet \\ M(p) + W(t, p) & \text{iff } p \in t \bullet / \bullet t \\ M(p) - W(p, t) + W(t, p) & \text{iff } p \in \bullet t \cap t \bullet \\ M(p) & \text{otherwise} \end{array} \right\}$$

► **Example A.2.3** By applying these firing rules to the initial marking of the dining philosopher net in the left-hand side of Figure A.1, 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. ♦

### A.2.2 High-level Petri nets (HLPN): An overview:

Basically, one of the main drawback of using petri-nets is the explosion of the number of the elements of their graphical form when they describe 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 (algebraic expression).

**Algebraic Petri nets** High-level Petri nets [JR91] and algebraic Petri nets [Rei91] 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 concise, 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 [Rei91]. 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  $SPEC = (S, OP, E)$ <sup>1</sup>, we denote the specification  $\mathbf{m\_SPEC}$  by  $(\widehat{S}, \widehat{OP}, \widehat{E})$ , respectively. Following the OBJ notation,  $\mathbf{m\_SPEC}$  can be described as follows:

```
obj m_SPEC is
  extending SPEC .
  sort m_s .
  op  $\vartheta_s$  :  $\rightarrow m_s$  .
  op MAKEs :  $s \rightarrow m_s$  .
  op  $-+_s-$  :  $m_s m_s \rightarrow m_s$  .
  op  $-_s-$  :  $m_s \rightarrow m_s$  .
  var  $t_1, t_2, t_3$  :  $m_s$ 
  eq  $t_1 +_s \vartheta_s = t_1$ 
  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  $t_1 +_s (-_s t_2) = \vartheta_s$ 
endo.
```

For sake of simplicity, in the following with multiset terms we will drop the sort indices  $s$  of operations symbols, and write  $\vartheta$  instead of  $\vartheta_s$ . 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  $\vartheta$ ,  $MAKE_s$  and  $+_s$ . This motivates the following concepts.

**Definition A.2.4 (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  $Sig = (S, OP)$ .

- (i) A mapping  $s : P \rightarrow S$  is called a *sort assignment* of  $N$ . Assuming  $s$ , for places  $p \in P$  let  $\tilde{p}$  denote the multiset sort  $m_{s(p)}$ .
- (ii) A mapping  $M_0 : P \rightarrow T_{OP^+}$  with  $M_0(p) \in T_{OP^+, \tilde{p}}$  for each  $p \in P$  is called a *s-respecting initial marking* of  $N$ .
- (iii) A mapping  $\lambda : F \rightarrow T_{OP^+}(X)$  with  $\lambda(f) \in T_{OP^+, \tilde{p}}(X)$  for each  $f = (t, p)$  or  $f = (p, t)$  is called a *s-respecting arc inscription* of  $N$ .
- (iv) A triple  $ins = (s, M_0, \lambda)$  of a sort assignment  $s$  of  $N$ , a *s-respecting initial marking*  $M_0$  of  $N$ , and a *s-sorted arc inscription*  $\lambda$  of  $N$ , is called a *SPEC-inscription* of  $N$ , and  $(N, ins, E)$  is *SPEC-inscribed net*. As a shorthand,  $N$  is said to be *inscribed* assuming that  $ins$  and  $E$  can be understood from the context.

► **Example A.2.5** (The dining philosophers as an algebraic net) A simple look at the dining philosophers modelling in Figure A.1 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 A.2 achieves such a

<sup>1</sup>We are using  $(S, OP, E)$  instead of  $(S, F, E)$  as previous due to the use of  $F$  as arc relation.

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 A.2 , 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  $s(\text{P\_eating})= s(\text{P\_thinking}) = \textit{phils}$ , and  $s(\text{Forks}) = \textit{forks}$ .  $M_0$  and  $\lambda$  are directly depicted in Figure A.2.

```

obj Phil is
  sort phils forks .
  op f1, f2, f3, f4, f5 : → forks .
  op p1, p2, p3, p4, p5 : → phils .
  op Lf : phils → forks .
  op Rt : phils → forks .
  var p_i, x : phils, f_i : forks
  eq Rt(p_i) = f_i, for i ∈ {1, ..., 5}
  eq Lf(p_i) = f_{i-1}, for i ∈ {1, ..., 5} with f_0 = f_5
endo.

```

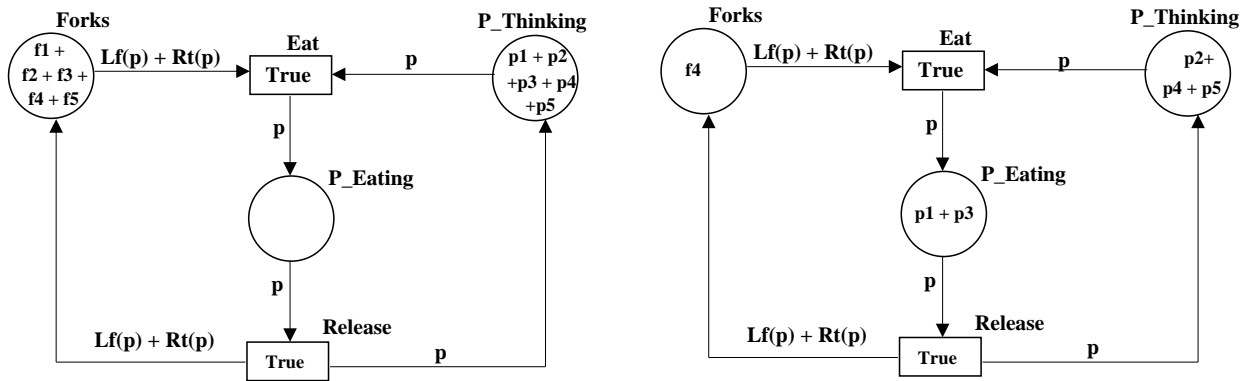


Figure A.2: The dining philosopher problem as an algebraic Petri net



► **Example A.2.6** In the right-hand side of Figure A.2 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.  $f_2 + f_3$ ). ◆

### A.2.3 Object-oriented Petri Nets: An overview

Important to point out at this level is that with the emerging of the object-oriented(OO) paradigm, several variants of high-level (OO) Petri nets have been proposed. Among these OO Petri Nets

formalisms, which allow to integrate different OO mechanisms (e.g. classification, inheritance, object-composition, etc), we may mention the following.

Cooperative Objects (CO) [SB94] is a formalism that aims at modeling an information system as a collection of objects that cooperate concurrently. Each cooperative object belong to a class. A cooperative object posses a type (the one of the class it belongs), an identity, and a state composed of values of built-in data types and/or references on the cooperative objects. Object Petri Nets (OPN) [Lak96] are presented as an extension of Coloured Petri Nets (CPN) [Jen92] which integrate object-oriented structuring concepts. The components of a net, whether tokens, places, transitions, or event subnets, now become objects. Each Petri net can be defined as a class which can be, as usual, instantiated. In addition to places and transitions, a class contains data fields and functions. CO-OPNET(concurrent Object-Oriented Petri Nets)[BBG97] is a specification language designed for the specification and the modeling of large concurrent systems.

Capitalizing on the strengths of such proposals, we put forwards in [AS02, AS04] a new form of component-based Petri nets for developing evolving concurrent information systems. The two underlying formalisms of CO-NETS are (order-sorted) algebraic specifications and Petri nets, while promoting intra- and inter-component interactions. The formalism is semantically governed by a rewriting logic-based theory.

### A.3 Rewriting techniques

Rewriting is an outgrowth of equational reasoning where instead of substituting equals by equals we substitute expressions by *simpler* expressions. Term rewriting systems have been widely proposed as computational substitutes for equational logic. Their main use has been in prototyping algebraic specifications of abstract data types.

We devote this subsection to a survey of some standard definitions and results about rewriting that are useful for the subsequent work. Our survey is more inspired by [?]. In the following definitions we assume that  $S$  is a set of sorts and  $Sig$  is an  $S$ -sorted signature.

**Definition A.3.1 (Sig-rewrite rule):** A *Sig-rewrite rule* (or simply rewrite rule if the signature is understood from the context) is a triplet  $(X, l, r)$  where  $X$  is a set of variables, and  $l$  and  $r$  are terms of some sort  $s$  with variables from  $X$  (i.e.  $l, r \in T_{Sig}^S(X)$ ). Generally, it is also required that  $var(r) \subseteq var(l)$  and no left-hand side can be a single variable. A term rewriting system, or TRS is a set of rewrite rules.

We usually write the rule  $(X, l, r)$  as  $(\forall X)l \Rightarrow r$ . The main difference between rewrite rules and equations is that the rewrite rules are directional. We will see below how this is reflected in the difference between the way rules and equations are used.

► **Example A.3.2** The following two rewrite rules allow reducing any natural number containing addition operator ‘+’ to a corresponding term with just the successor operator  $s$ , as natural numbers constructor.

```

obj NAT is
  sort Nat .
  op 0 : → Nat [ctor] .
  op s _ : Nat → NzNat .
  op _ + _ : Nat Nat → Nat [assoc comm].
  var N : Nat .
  var M : Nat .
  rl N + 0 ⇒ N .
  rl s(N) + M ⇒ s(N + M) .
endo

```

For the introduction of the (rewrite) relation induced by a rewriting system, we need the definition of term positions and replacement at a given position.

**Definition A.3.3 (Term-Position):** Given a term  $t$ , the set of *position* in  $t$ , denoted by  $Dom(t)$ , is the set of sequences of natural numbers defined as:

- If  $t$  is a constant or a variable, then  $Dom(t) = \{\phi\}$ .
- If  $t$  is of the form  $f(t_1, \dots, t_n)$ , then  $Dom(t) = \{\phi\} \cup \{i.p \mid i \in \{1, \dots, n\} \wedge p \in Dom(t_i)\}$ .  $\square$

Term positions associated with a term are usually depicted as a tree, there each node represents a position.

► **Example A.3.4** With respect to the above example, let  $t$  be the term  $t = ss0 + (0 + s0)$ ; the corresponding tree is displayed in Figure ?? with

- $Dom(t) = \{\phi, 1, 2, 1.1, 2.1, 2.2, 1.1.1, 2.2.1\}$

This example shows that each  $p \in Dom(t)$  corresponds, in the associated tree, to a "path" from the root to some node.

**Definition A.3.5 (Subterms)** Given a term  $t$ , and a position  $p \in Dom(t)$ , we define the *subterm* of  $t$  rooted at  $p$ , denoted by  $t|_p$ , as:

- If  $p = \phi$  then  $t|_p = t$
- If  $p = i.p'$  (and therefore  $t$  is of the form  $f(t_1, \dots, t_{i-1}, t_i, t_{i+1}, \dots, t_n)$  for some  $n \geq i$ ) then  $t(t_1, \dots, t_n)|_{i.p'} = t_i|_{p'}$ .
- A term  $t'$  is said to be a subterm of  $t$  iff there exists  $p \in Dom(t)$  such that  $t' = t|_p$ .

► **Example A.3.6** From the term  $t = ss0 + (0 + s0)$  in the above example, we have for instance:  $t|_1 = ss0$ , and  $t|_{2.2} = s0$ .

**Definition A.3.7 (Term replacement):** Given a term  $t \in T_{Sig,s}$ , a position  $p \in Dom(t)$  such that  $t|_p \in T_{Sig,s'}$ , and a term  $t' \in T_{Sig,s'}$  we define  $t[p \leftarrow t']$  as

- If  $p = \phi$  then  $t[p \leftarrow t'] = t'$

- If  $p = i.p'$  (and therefore  $t$  is of the form  $f(t_1, \dots, t_{i-1}, t_i, t_{i+1}, \dots, t_n)$  for some  $n \geq i$ ), then  $f(t_1, \dots, t_{i-1}, t_i, t_{i+1}, \dots, t_n)[i.p \leftarrow t'] = f(t_1, \dots, t_{i-1}, t_i[p \leftarrow t'], t_{i+1}, \dots, t_n)$ .

► **Example A.3.8** From the term  $t = ss0 + (0 + s0)$  in example 2.2.2, we have for instance:  $t[1 \leftarrow 0] = 0 + s0$

**Definition A.3.9 (Rewriting a term):** Given a TRS  $\mathcal{R}$ , we define a *rewrite* relation over  $T_{Sig}(Y)$ , denoted by  $\Rightarrow_{\mathcal{R}}$ , as  $t \Rightarrow_{\mathcal{R}} t'$  if and only if there exist

- a rule  $(\forall X) \quad l \Rightarrow r$  in  $\mathcal{R}$ , where  $t$  and  $r$  are of some sort  $s$ ,
- a substitution  $\sigma : X \rightarrow T_{Sig}(Y)$ ,
- a position  $p$  in  $t$  such that  $t|_p$  is of sort  $s$ , such that:  $t|_p = \bar{\sigma}(l)$  and  $t' = t[p \leftarrow \bar{\sigma}(r)]$ .

In this case we say that  $t$  rewrites (in one step) into  $t'$  at position  $p$ . We speak about concurrent rewriting when this rewriting process is applied in parallel to several (independent) positions.

► **Example A.3.10** Let  $t$  be the term  $t = (ss0 + s0) + s(0 + s0)$ . In this term the subterm at position 1, that is  $t_1 = ss0 + s0$  can be matched with the left-hand term of the second rewrite rule in example 2.2.2 (i.e.  $s(m) + n \Rightarrow s(m+n)$ ). The corresponding substitution is  $\bar{\sigma} = \{m \rightarrow s0, n \rightarrow s0\}$ . The term  $t$  can be rewritten to the more simplified form  $t' = s(s0 + s0) + s(0 + s0)$ . Note that this term can be simultaneously rewritten at positions 1 and 2. In this case the term  $t$  becomes  $t' = s(s0 + s0) + ss0$ . More applications of both rules result in the simplified term  $t = sss0$  (also called in a "normal form").

To result in a decision procedure based on rewriting techniques for the equational logic, two properties are required for a given rewriting system: the *termination* and *confluence*. While the termination should ensure that any rewriting process terminates, the confluence property ensures that all non-deterministic rewritings of a given term result in the same (simplified) term called normal form.

## A.4 Rewriting logic

Rewriting logic, as a new paradigm for concurrent systems, has been introduced by J. Meseguer in [Mes92] by observing, first, that *concurrent* rewriting is a natural process in term rewriting and second the inadequacy of interpreting rewrite rules as (oriented) equations when dealing with non Platonic (i.e. reactive) systems. While rewrite rules have the usual form, they are rather interpreted in rewriting logic as a *change* in concurrent systems. In this sense rewriting logic has been proved as an elegant and expressive semantic framework for the specification of languages and systems, and it is a good candidate as a logical framework in which many other logics can be represented [MOM96].

### A.4.1 Rewriting Logic and its Theory

A signature in rewriting logic is a pair  $(\Sigma, E)$  with  $\Sigma$  a ranked alphabet of function symbols and  $E$  a set of  $\Sigma$ -equations. Rewriting will operate on equivalence classes of terms modulo the set of equations  $E$ . In this way, we free rewriting from the syntactic constraints of a term representation and gain a much greater flexibility in deciding what counts as a *data structure*; for example, string rewriting is obtained by imposing an associativity axiom, and multiset rewriting by imposing associativity and commutativity. Of course, standard term rewriting is obtained as the particular case in which the set  $E$  of equations is empty. To be more precise we present some definitions borrowed from [?]:

**Definition A.4.1 (Rewrite Theory):** A (*labelled*) *rewrite theory*  $\mathcal{R}$  is a 4-tuple  $\mathcal{R} = (\Sigma, E, L, R)$  where  $\Sigma$  is a ranked alphabet of functions symbols,  $E$  is a set of  $\Sigma$ -equations,  $L$  is a set called the set of *labels* and  $R$  is a set of pairs  $R \subseteq L \times (T_{\Sigma, E}(X)^2)^+$  whose first component is a label, and whose second component is a nonempty sequence of pairs of  $E$ -equivalence classed of terms, with  $X = x_1, \dots, x_n$  a countably infinite set of variables. Elements of  $R$  are called *rewrite rules*. A rewrite rule  $(r, [t], [t'])([u_1], [v_1]) \dots ([u_k], [v_k])$  is denoted as

$$r : [t] \Rightarrow [t'] \quad \text{if} \quad [u_1] \Rightarrow [v_1] \bigwedge \dots \bigwedge [u_k] \Rightarrow [v_k]. \quad \square$$

**Definition A.4.2 (Rewriting entailment inference rules)** Given a rewrite theory  $\mathcal{R}$ , we say that  $\mathcal{R}$  *entails* a sequent  $r : [t] \Rightarrow [t']$  and write  $\mathcal{R} \vdash [t] \Rightarrow [t']$  *if*  $[t] \Rightarrow [t']$  can be obtained by finite application of the following *rules of deduction*:

- (1) **Reflexivity** : For each  $[t] \in T_{\Sigma, E}(X)$ ,

$$\frac{}{[t] \Rightarrow [t]}$$

- (2) **Congruence** : For each  $f \in \Sigma_n, n \in \mathbb{N}$

$$\frac{[t_1] \Rightarrow [t'_1] \dots [t_n] \Rightarrow [t'_n]}{[f(t_1, \dots, t_n)] \Rightarrow [f(t'_1, \dots, t'_n)]}$$

- (3) **Replacement** : For each rule  $r : [t(x_1, \dots, x_n)] \Rightarrow [t'(x_1, \dots, x_n)]$  in  $R$ ,

$$\frac{[w_1] \Rightarrow [w'_1] \dots [w_n] \Rightarrow [w'_n]}{[t(\vec{w}/\vec{x})] \Rightarrow [t(\vec{w}'/\vec{x})]}$$

- (4) **Transitivity** :

$$\frac{[t_1] \Rightarrow [t_2] \quad [t_2] \Rightarrow [t_3]}{[t_1] \Rightarrow [t_3]}$$

**Definition A.4.3 (Concurrent rewriting):** Given a rewrite theory  $\mathcal{R} = (\Sigma, E, L, R)$ , a  $(\Sigma, E)$ -sequent  $[t] \Rightarrow [t']$  is called:

- a *0-step concurrent  $\mathcal{R}$ -rewrite* iff it can be derived from  $(\mathcal{R})$  by finite application of the rules (1) and (2) of rewriting deduction (in which case  $[t]$  and  $[t']$  necessarily coincide);

- a *one-step concurrent  $\mathcal{R}$ -rewrite* iff it can be derived from  $(\mathcal{R}$  by finite application of the rules (1)-(3), with at least one application of rule (3); if rule (3) was applied exactly once, we then say that the sequent is a *one-step sequential  $\mathcal{R}$ -rewrite*.
- a *concurrent  $\mathcal{R}$ -rewrite* (or just a *rewrite*) iff it can be derived from  $\mathcal{R}$  by finite application of the rules (1)-(4).

We call the rewrite theory  $\mathcal{R}$  *sequential* if all one-step  $\mathcal{R}$ -rewrites are necessarily sequential. A sequential rewrite theory  $\mathcal{R}$  is in addition called *deterministic* if for each  $[t]$  there is at most one one-step (necessary sequential) rewrite  $[t] \Rightarrow [t']$ .  $\square$

We also point out that the practicability of this logic is enhanced by the development of an adequate language called MAUDE [CDE<sup>+</sup>07]. In MAUDE besides usual functional specifications (i.e. algebraic modules), the so-called system or object-oriented modules can be specified and semantically interpreted as theories in rewrite logic. In the next chapter we will discuss about the capabilities of this language.

#### A.4.2 The meaning of Rewriting Logic

In rewriting logic a sequent  $[t] \Rightarrow [t']$  should not be read as "[t] equals [t']", but as "[t] becomes [t']". Clearly, rewriting logic is a logic of *becoming* or *change*, not a logic of equality in a static Platonic sense. The apparently innocent step of adding the symmetry rule is in fact a *very strong* restriction, namely assuming that *all changes is reversible*, thus bringing us into a timeless platonic realm in which "before" and "after" have been identified.

A related observation is that  $[t]$  should not be understood as a *term* in the usual first-order logic sense, but as a *proposition*—built up using the *propositional connectives* in  $\Sigma$ —that asserts being in a certain *state* having a certain *structure*. However, unlike most other logics, the logical connectives  $\Sigma$  and their structural properties  $E$  are entirely *user-definable*. This provides great flexibility for considering many different state structures and makes rewriting logic very general in its capacity to deal with many different types of concurrent systems.

### A.5 MAUDE and its Reflection : Overview

To make this thesis self-contained, we introduce main principles around this language. That is, first we recall the main features of the MAUDE language. Then, since all component datatypes are described in MAUDE as functional modules, we review this functional-level. For specifying object-oriented applications, we then introduce all the ingredients underlying MAUDE system and object modules. Finally, we present how to control the rewriting rules using reflection in general. More specifically, we present how internal strategies can be specified in the MAUDE language.

## A.6 MAUDE main Features

MAUDE is a high-level language and high-performance system supporting both functional and object-oriented specifications and programming for a wide range of applications. MAUDE has been influenced in important ways by the functional OBJ3 language [?]. The main features and characteristics of the MAUDE language, could be summarized as follows:

**MAUDE is rewriting logic-based** Rewriting logic [Mes92] is a logic of concurrent changes that can deal with state and with highly nondeterministic concurrent computations. This makes it particularly well suited to express in a declarative way concurrent and state-changing aspects of systems. MAUDE programs are theories, and rewriting logic deduction exactly corresponds to concurrent computation.

**Wide-spectrum based-logic.** Rewriting logic is a flexible and general semantic framework for a wide range of languages. At the logical level, MAUDE allows [?, ?] executable specifications, rapid prototyping, and efficient parallel and distributed executions.

**MAUDE is Reflective** Rewriting logic and MAUDE are reflective [CM96]. That is, they are able to express their own metalevel at the object level. The design of MAUDE capitalizes on this fact to support a novel style of metaprogramming, which includes both user-definable module operations and declarative strategies to guide the deduction process. It offers thus very powerful module-combining and module-transforming operations that surpass those of traditional parameterized programming. This can greatly advance software reusability and adaptability. The MAUDE strategies for controlling the rewriting process are defined by rewrite rules at the metalevel and can be reasoned about inside the logic. Therefore, instead of having a "Logic + Control" introduction of extra-logical features, in MAUDE "Control  $\subseteq$  Logic".

MAUDE modules are rewriting theories, while computation with such modules corresponds to efficient deduction by rewriting. There are three types of modules in MAUDE: Functional module (fmod), the system module (mod), and the object-oriented module (omod).

MAUDE's functional modules are theories in membership equational logic, which extends order-sorted equational logic and supports sorts, subsort relations, operator overloading, definition of partial functions with equationally defined domains, and error specification. They are assumed to be Church-Rosser and termination. Membership equational logic is a sublogic of rewriting logic [?].

MAUDE's system modules are rewrite theories, in which the local transition rules in a concurrent system, or the inference rules in a logical system, instead of equations as the rewrite rules. In rewrite logic, the rewrite rules need not be terminating and Church-Rosser.

In addition, MAUDE supports rewriting modulo equational theories such as associativity, commutativity, and identity. Therefore, we can not only have infinite chains of rewriting, we may but also have highly divergent rewriting paths, which could never cross each other by further rewriting.



Hence, we need to have good ways of controlling the rewriting inference process. Using reflection, the rewriting inference process can be controlled with great flexibility in MAUDE by means of internal strategies. This chapter explains and illustrates with examples the main concepts of such MAUDE's language concepts.

### A.6.1 MAUDE Functional Modules

Functional modules define data types and operations on them by means of equational theories. Computation in a functional module is accomplished by using the equations as rewrite rules. That is, each step of rewriting is a step of replacement of equals by equals, until a canonical form is found. For this reason, the equations in the functional module must satisfy the additional requirements of being Church-Rosser, terminating, and sort decreasing.

The equational logic on which MAUDE functional modules are based is an extension of order-sorted equational logic called membership equational logic [?]. In addition to supporting sorts, subsort relations, and overloading of function symbols, functional modules also support membership axioms, a generalization of sort constraints in which a term is asserted to have a certain sort if a condition consisting of a conjunction of equations and unconditional membership tests is satisfied. Such membership axioms can be used to define partial functions, that become defined when their arguments satisfy certain equational and membership conditions.

We can illustrate these ideas, as well as MAUDE's support for mixfix user-definable syntax, with the following MAUDE functional module *NAT*:

```

1.fmod NAT is
2.sorts NzNat Nat .
3.subsort NzNat < Nat .
4.op 0 : → Nat [ctor] .
5.op s _ : Nat → NzNat .
6.op _ + _ : Nat Nat → Nat [assoc comm].
7.var N : Nat .
8.var M : Nat .
9.cmb N / M : NzNat if (N ≠ 0) .
10.eq N + 0 = N .
11.eq s(N) + M = s(N + M) .
12.endfm

```

The modules are introduced with the functional module syntax *fmod ... endfm* and have names, *NAT*. The statement *protecting* imports module. The sorts and subsort relations of this module are introduced by a sort and a subsort declarations by the keyword `sort(s)` and `subsort(s)`. Sorts are used to classify data. A subsort relation between two sorts is interpreted as a set-theoretic inclusion, that is, it means that the data of the subsort is included in that of the supersort.

**Membership** simply refers to how certain terms are "members" of sorts. When we declare a variable, we declare it as a member of a sort using the colon, which one can think of a symbol for "is a member of". Thus, the declaration `var N : Nat` is the same as saying "the variable `N` is a member of the sort `Nat`".

### A.6.2 System and object-oriented Modules

The passage from functional modules to system modules involves a fundamental change in perspective, so that basic notations that previously had a very familiar interpretation in functional

terms have now to be reinterpreted. As mentioned already, in this new interpretation, a term  $t$  is no longer understood as a functional expression, but as a structured state of a system, where the structure of the state is given by the operators that happen to appear in the term and by structural axioms that they enjoy. The algebraic structure of the state—as a multiset, binary tree or whatever—is precisely what makes the state distributed, i.e., coincides with its distributed structure, and makes concurrency possible. In the same way, a rewrite rule  $t \rightarrow t'$  is no longer seen as functional evaluation by equational deduction, but as a local state transition, stating that if a portion of a system's state exhibits the pattern described by  $t$ , then that portion of the system can change to the corresponding instance of  $t'$ .

**MAUDE System Modules.** We can represent a rewrite theory as a 4-tuple  $\mathcal{R} = (\Omega, E, L, R)$  where  $(\Omega, E)$  is a theory in membership equational logic, that specifies states of the system as an abstract data type,  $L$  is a set of labels, to label the rules, and  $R$  is a set of labeled rewrite rules axiomatizing the local state transitions of the system with either of the unconditional form  $rl : [t] \rightarrow [t']$ , or of the conditional form  $crl : [t] \rightarrow [t']$  if  $C, C$  means the conditions connecting with  $\wedge$  which is associative.

The most general MAUDE modules are system modules. They specify the initial module of a rewrite theory  $(\Omega, E, L, R)$ , in which the signature  $\Omega$  is given by the sorts, subsort relations, and operator declarations, a set  $E$  of equations, that is assumed to be decomposed as a union  $E = A \cup E'$ , with  $A$  a set of axioms to rewrite module among those supported by MAUDE, and  $E'$  a set of Church-Rosser and termination equations module  $A$ .

**MAUDE: Object-oriented module.** To present a logical theory of concurrent objects based on rewriting logic deduction modulo *ACI* (*associativity, commutativity and identity*), the key idea is to conceptualize the distributed state of a concurrent object-oriented system—called a configuration. It is as a multiset made up of objects states and messages instances flowing together that evolved by concurrent rewriting modulo associativity, commutativity and identity, where the rules describes the effects of events between objects and messages. Therefore, we can view concurrent object-oriented computation as *deduction* in rewriting logic. In MAUDE, object states are conceived as tuples of the form

$$\langle Id : C | at_1 : v_1, \dots, at_k : v_k \rangle$$

where  $Id$  stands for the object identity,  $C$  for its class while  $atr_1, \dots, atr_k$  denote attribute identifiers with respective current values  $val_1, \dots, val_k$ . Messages can be concurrently sent / receive to such object states, and both object and message instances flow together in the so-called *configuration*, which introduces the basic concepts of concurrent object systems, as multiset governed by the union operator denoted by  $'_ \cdot '$ . The precise definition of this configuration in MAUDE itself takes the form.

```

mod Configuration is
  protecting ID      **** provides OId, CId and AId .
  sorts Configuration Object Msg .
  subsorts OId < Value .
  subsorts Attribute < Attributes .
  subsorts Object Msg < Configuration .
  op  $\_ : \_$  : AId Value  $\rightarrow$  Attribute .
  op  $\_ , \_$  : Attribute Attributes  $\rightarrow$  Attributes [assoc. comm. Id:nil]
  op  $\langle \_ : \_ \rangle$  : OId CId Attributes  $\rightarrow$  Object.
  op  $\_ \_$  : Configuration Configuration  $\rightarrow$  Configuration [assoc comm id:null].
endm

```

In MAUDE, concurrent object-oriented system can be defined by means of object-oriented modules—introduced by the keyword (`omod ... endom`)—using a syntax more convenient than that of system modules. This is because it assumes acquaintance with basic entities, such as objects, messages and configurations, and supports linguistic distinctions appropriate for the object-oriented case. In particular, all object-oriented modules implicitly include the above *CONFIGURATION* module and assume its syntax. Further, they are internally transformed into system modules for execution purposes (that is introduced in the next subsection).

As example, the following *ACCNT* object-oriented module specifies the usual concurrent behavior of banking accounts.

```

(omod ACCNT is
  protecting REAL
  class Accnt | bal : NNReal .
  msgs credit debit: OId NNReal  $\rightarrow$  Msg .
  msg transfer_from_to  $\_$  : NNReal OId OId  $\rightarrow$  Msg .
  vars A B : OId .
  Vars M N N': NNReal .
  ***** The Account behaviour.
  rl debit(A,M)  $\langle A : \text{Accnt} | \text{Bal} : N \rangle \Rightarrow \langle A : \text{Accnt} | \text{Bal} : N - M \rangle$  if  $N \geq M$  .
  rl credit(A,M)  $\langle A : \text{Accnt} | \text{Bal} : N \rangle \Rightarrow \langle A : \text{Accnt} | \text{Bal} : N + M \rangle$  .
  rl transfer M from A to B  $\langle A : \text{Accnt} | \text{Bal} : N \rangle \langle B : \text{Accnt} | \text{Bal} : N' \rangle \Rightarrow$ 
     $\langle A : \text{Accnt} | \text{Bal} : N - M \rangle \langle B : \text{Accnt} | \text{Bal} : N' + M \rangle$  if  $N \geq M$  .
endom)

```

Classes are defined with the keyword `class`, followed by the name of the class *C*, and by a list of attribute declarations separated by commas. Each attribute declaration has the form `a : S`, where `a` is an attribute identifier and *S* is the sort in which the values of the attribute range; that is, class declarations have the form `class C | a1 : S1, ..., an : Sn`. In this example, the account class (*Accnt*) has only one attribute `bal`, which is declared to be a value of type `NNReal` (non-negative real number).

The syntax for message declarations is similar to the syntax for the declaration of operators, using keywords `msg` and `msgs`, and having as result sort `Msg` or a subsort of it. In the above account example, the three kinds of messages—`credit`, `debit`, and `transfer`—are introduced by the keyword `msg` and their resulting sorts are `Msg`. The debit rule, for instance, says that when an account state receives a debit message, `debit(A, M)`, the next state results in decreasing the balance with the corresponding amount of money, and this under the condition that the current balance suffices.

The rewrite rules specify in a declarative way the behavior associated with the credit, debit, and transfer messages. The multiset structure of the configuration provides the top-level distributed structure of the system and allows concurrent application of the rules. To illustrate the above account module *ACCNT*, we propose the following simple account configuration *ACNT-CONF*, which consists of three accounts, two debit, two credit and one transfer. The rewrite rule executions are controlled by *default strategies*.

```
(omod ACNT-CONF is
  ex ACCNT
  ops A1 A2 A3 → Oid .
  op AcCfCp → Configuration .
  eq AcCfCp = < A1 : Accnt | Bal : 500 > debit(A1, 200) < A2 : Accnt | Bal : 100 > credit(A2, 50)
              credit(A1, 300) debit(A2, 300) < A3 : Accnt | Bal : 300 > (transfer 200 from A1 to A3) .
  endom)
```

Below we show the rewrite of the instances and the results. Figure A.3 provides the snapshots in the evolution by concurrent rewriting on this example.

```
MAUDE> rew in ACNT-CONF : AcCfCp .
ResultObject :< A1 : Accnt | Bal : 400 >< A2 : Accnt | Bal : 150 >< A3 : Accnt | Bal : 500 > debit(A2, 300)
```

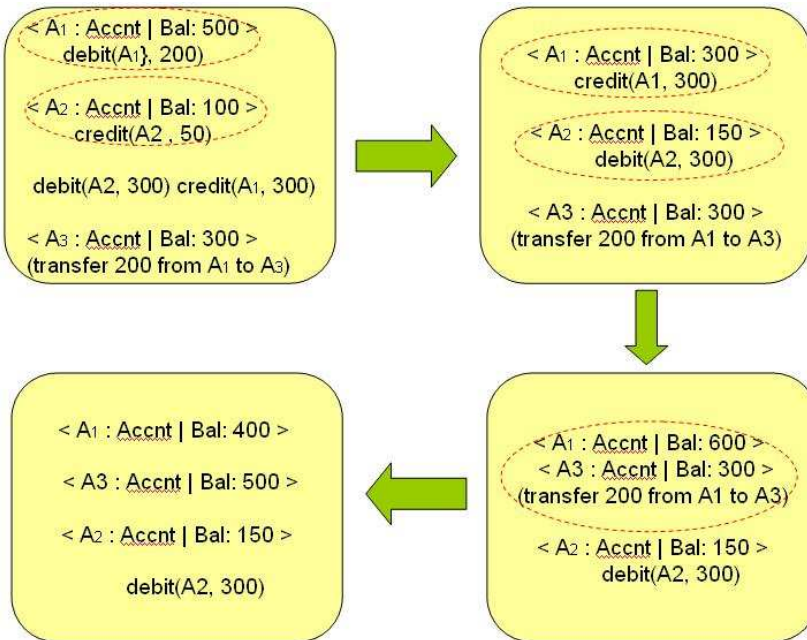


Figure A.3: Concurrent rewriting of bank accounts.

### A.6.3 MAUDE Reflection and internal Strategies

Informally, a reflective logic is a logic in which important aspects of its metatheory can be represented at the object level in a consistent way, so that the object-level representation correctly

simulates the relevant metatheoretic aspects. In other words, a reflective logic is a logic which can be faithfully represented in itself.

Rewriting logic is reflective [MOM96] by essence. That is, any rewrite theory can be (meta-)represented at a higher level and be reasoned on like data manipulation. This reflection property has been nicely defined using the following abstraction, where universal  $\mathcal{U}$  represents the (meta-)theory at the higher level in which any rewrite system and / or terms can be manipulated as (meta-)data.

$$(†) \mathcal{R} \vdash t \longrightarrow t' \iff \mathcal{U} \vdash \langle \overline{\mathcal{R}}, \bar{t} \rangle \longrightarrow \langle \overline{\mathcal{R}}, \bar{t}' \rangle$$

*META-LEVEL* has sorts *Term* and *Module*, so that the representations of a term  $t$  and of a module  $\mathcal{R}$  are, respectively, a term  $\bar{t}$  of sort *Term* and a term  $\overline{\mathcal{R}}$  of sort *Module*.

The module *META-LEVAL* also provides key functions for rewriting and evaluating terms at the meta-level, namely, `upModule`, `upTerm`, `downTerm`, `metaReduce`, `metaRewrite`, `metaApply`, `metaXapply`, etc.

**Moving between reflection levels:** `upModule`, `upTerm` and `downTerm`. The operation `upModule` takes as arguments the metarepresentation of the name of  $\mathcal{R}$  and a Boolean value  $b$ , and returns, respectively, the metarepresentation of the modules  $\mathcal{R}$ . The polymorphic functions `upTerm` and `downTerm` can move terms between the reflection levels. `upTerm` transfers a term into its metarepresentation. To display the output in a more readable form we can use `downTerm` function, which in a sense inverse to `upTerm`, since it gives us back the term from its metarepresentation.

```
op upTerm : Universal → Term [poly special (...)] .
op downTerm : Term Universal → Universal [poly special (...)] .
```

**Simplifying:** `metaReduce` and `metaRewrite`. The function `metaReduce` takes as arguments the metarepresentation of a module  $R$  and the metarepresentation of a term  $t$  in that module, and returns the metarepresentation of the fully reduced form of the term  $t$  using the equations in  $R$ , together with its corresponding sort or kind:

```
op metaReduce : Module Term ~> ResultPair [special (...)] .
op {_, _} : Term Type → ResultPair [ctor] .
```

The (partial) operation `metaRewrite` is entirely analogous to `metaReduce`, but uses both the equations and the rules to rewrite the term. The function `metaRewrite` takes as arguments the metarepresentation of a module  $R$ , the metarepresentation of a term  $t$ , and a value  $b$  of the sort `Bound`, i.e., either a natural number or the constant `unbounded`.

**Applying rules:** `metaApply` and `metaXapply`. The `metaApply` basically takes a term and a rewrite law in a given module, and then rewrites the term once by applying the specified law. The operation `metaXapply` has syntax:

```

op metaApply : Module Term Qid Substitution Nat ~> ResultTriple?
                [special (...)] .
op {- , - , -} : Term Type Substitution → ResultTriple [ctor] .

```

The first argument `Module` is the meta-representation of the module that defines the terms and laws in question, the `Term` is the given term to be rewritten, that must match the left-hand side of the rewrite rule applied, and match it exactly (modulo associativity, commutativity, etc.) and the `Qid` is the name of the rewrite law to be applied.

The operation `metaXapply` applies a rule on a term in any possible position. The first four arguments are the metarepresentation of a module `R`, the metarepresentation of a term `t` in `R`, a label `l` of some rules in `R`, and a set of assignments (possibly empty) defining a partial substitution  $\sigma$  for the variables in those rules.

```

op metaXapply : Module Term Qid Substitution Nat Bound Nat ~>
                Result4Tuple? [special (...)] .
op {- , - , - , -} : Term Type Substitution Context → Result4Tuple
                [ctor] .

```

`metaXapply` returns a tuple of sort `Result4Tuple` consisting of a term, with the corresponding sort or kind, a substitution, and the context inside the given term where the rewriting has taken place.

#### A.6.4 Internal Strategies

As mentioned already, system modules in MAUDE are rewrite theories that do not need to be Church-Rosser and terminating. Therefore, we need to have good ways of controlling the rewriting inference process by means of adequate strategies. Using reflection the rewriting inference process can be controlled with great flexibility in MAUDE by means of internal strategies that can be defined using statements in a normal module in MAUDE, and can be reasoned about as with statements in any other module. In fact, there is great freedom for defining many different types of strategies, or even many different strategy languages inside MAUDE. This can be done in a completely user-definable way, so that users are not limited by a fixed and closed particular strategy language. In general, strategies for controlling the application of the rules are defined in extensions of the META-LEVEL module by using `metaReduce`, `metaApply`, `metaXapply` etc., as building blocks, which are then combined to obtain more complex strategies.

To illustrate the possibilities by implementing the following strategies for controlling the execution of the rules in the account specification ACCNT that is defined in the section 3.3.2 : first all deposits that appear in instance are performed, then withdrawals and finally all transfers, we propose a very basic strategy module *ACCNT-STR* below, which imports the META-LEVEL module.

```

mod ACCNT-STR is
inc ACNT-CONF .
protecting META-LEVEL .
vars debit? credit? transfer? : [Result4Tuple] .
var T : Term .
op Compute : Term → Term .
ceq Compute(T)

```

```

= (if(debit? :: Result4Tuple)
  then getTerm(debit?)
  else if(transfer? :: Result4Tuple)
  then getTerm(transfer?)
  else if(credit? :: Result4Tuple)
  then getTerm(credit?)
  else T fi fi fi)
if debit? = metaXapply(upModule('ACCNT, false), T,
  'debit, none, 0, unbounded, 0)
∧ credit? = metaXapply(upModule('ACCNT, false), T,
  'credit, none, 0, unbounded, 0)
∧ transfer? = metaXapply(upModule('ACCNT, false), T,
  'transfer, none, 0, unbounded, 0) .
endm

```

We still use the instances that we have introduced. Below we show the rewrite of the instances in the META-LEVEL and the results, and Figure A.4 provides the snapshots of explicitly controlling of the executions of different rules in the instance by our strategies.

```

MAUDE> reduce in ACNT-STR : downTerm(Compute(upTerm(AcCfCp)),
  'error) .
ResultObject : < A1 : Accnt | Bal : 400 > < A2 : Accnt | Bal : 150 >
  < A3 : Accnt | Bal : 500 > debit(A2, 300) .

```

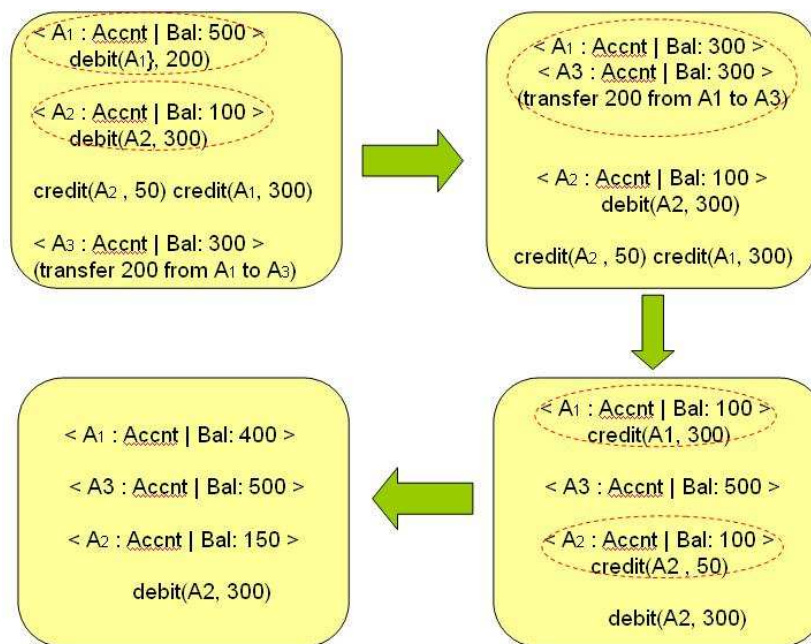


Figure A.4: Strategies control the rules execution.

### A.6.5 MAUDE-Workstation : presentation

MAUDE Workstation is a programming environment for MAUDE. It's written in Java what makes it executable in different platforms. In this paper the MAUDE modules are implemented in this MAUDE-Workstation environment, whose general view is depicted in Figure A.5.

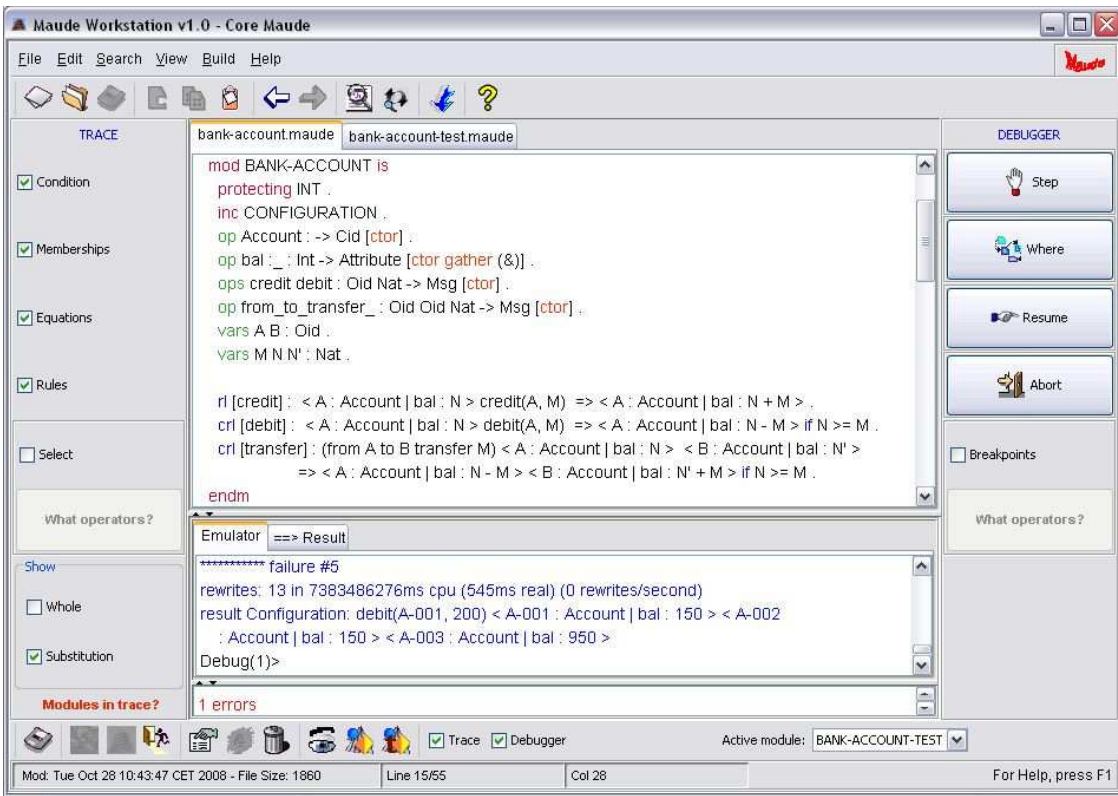


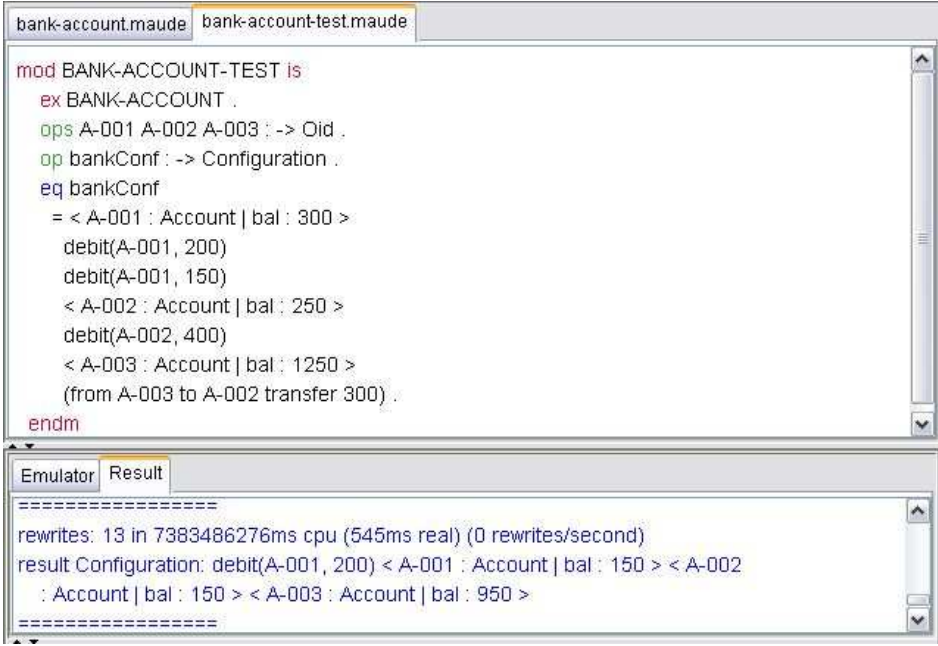
Figure A.5: General view of MAUDE Workstation.

The environment has two main parts: the edition facilities and the MAUDE emulation area. The first part is located in the upper area, and the second one in the lower one. In the edition area, the different opened files are accessible through a split pane system.

There are also two tool bars (the trace one on the left and the deputation one on the right) with show up whenever the user needs. They are located in vertical position on both sides of our environment. As a result, we can also manipulate very easily tracing and deputation facilities.

MAUDE Workstation also has a sequence of tool bars that shows full information on the specifications of Core and Full MAUDE. This tool is the window Show information. With this window we can recover the information from the modules stored in the local database without having to communicate with the MAUDE process.





The screenshot displays the MAUDE Workstation interface. At the top, there are two tabs: "bank-account.maude" and "bank-account-test.maude". The main window is split into two sections. The upper section contains a code editor with the following text:

```
mod BANK-ACCOUNT-TEST is
  ex BANK-ACCOUNT .
  ops A-001 A-002 A-003 : -> Oid .
  op bankConf : -> Configuration .
  eq bankConf
    = < A-001 : Account | bal : 300 >
      debit(A-001, 200)
      debit(A-001, 150)
    < A-002 : Account | bal : 250 >
      debit(A-002, 400)
    < A-003 : Account | bal : 1250 >
      (from A-003 to A-002 transfer 300) .
  endm
```

The lower section is a split panel with two tabs: "Emulator" and "Result". The "Result" tab is active and displays the following output:

```
=====  
rewrites: 13 in 7383486276ms cpu (545ms real) (0 rewrites/second)  
result Configuration: debit(A-001, 200) < A-001 : Account | bal : 150 > < A-002  
: Account | bal : 150 > < A-003 : Account | bal : 950 >  
=====
```

Figure A.6: The result split panel of MAUDE Workstation.

## Appendix B

# N.Aoumeur Publications Related to this Thesis

- [ABS09k] N. Aoumeur, K. Barkaoui, and G. Saake. Stepwise Engineering and Deployment of Dynamically Adaptive Service-oriented Business Processes. In *Proc. of IEEE International Conference on Service-Oriented Computing and Applications (SOCA'09)*, to appear, Dec. 14-15. IEEE CS Press, 2009.
- [ABS09a] N. Aoumeur, K. Barkaoui, and G. Saake. A Multi-Dimensional Architectural Approach to Behavior-Intensive Adaptive Pervasive Applications. In *Proc. of 4th International Symposium on Wireless Pervasive Computing (ISWPC'09)*, pages 1–8. IEEE CS Press, 2009.
- [ABS09b] N. Aoumeur, K. Barkaoui, and G. Saake. On Agile Service-oriented Business Processes: Activity-centric ECA-Architectural Foundation with Aspectual .NET Environment. In *Proc. of 4th I1st Workshop sur les Services Web dans les Systemes D'Information (WWS'09)*, pages 1–13. IEEE CS Press, 2009.
- [ABS09c] N. Aoumeur, K. Barkaoui, and G. Saake. Rapid-prototyping of Adaptive Component-based Systems using Runtime Aspectual Interactions. In *Proc. of 20th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP'09)*, pages 18–25. IEEE CS Press, 2009.
- [ABS09d] N. Aoumeur, K. Barkaoui, and G. Saake. Towards a Disciplined Engineering of Adaptive Service-oriented Business Processes. In *Proc. of 4th IEEE International Conference on Internet and Web Applications and Services (ICIW'09)*, pages 474–480. IEEE CS Press, 2009.
- [ABS09e] N. Aoumeur, K. Barkaoui, and G. Saake. Validating, Composing and Dynamically Adapting Features in Concurrent Product-Lines Applications. In *Proc. of 16th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS'09)*. IEEE CS Press, 2009. To appear.

- 
- [AFO04a] N. Aoumeur, J. Fiadeiro, and C. Oliveira. Distribution Concerns in Service-Oriented Modelling. In *2nd International Conference on Service Oriented Computing (ICSOC'04)*, pages 26–35. ACM Press, 2004.
- [AFO04b] N. Aoumeur, J. Fiadeiro, and C. Oliveira. Towards an Architectural Approach for Location-aware Business Processes. In *Proc. of the 13th IEEE International Workshops on Enabling, Technologies : Infrastructure for Collaborative Enterprises, June 14-16*, pages 147–152. IEEE Computer Society, 2004.
- [AFO06] N. Aoumeur, J. Fiadeiro, and C. Oliveira. Distribution Concerns in Service-Oriented Modelling. *International Journal of Internet Protocol Technology (IJPT)*, 1(3):144–158, 2006.
- [AGR08] N. Aoumeur, G. Saake, and C. Rautenstrauch. On Adaptive and Behavioral Service-Driven Applications: A Rule-Centric Petri Nets Framework. In *Proc. of the 4th International Conference on Signal-Image Technology & Internet-based Systems (SITIS'08)*, pages 195–202. IEEE CS Press, 2008.
- [RLA+09] S. Rahman, A. Lodhi, N. Aoumeur, C. Rautenstrauch, and G. Saake. Intra-Service Adaptability for ECA-Centric Web Services using Contract and Aspect. In *Proc. of IADIS International Conference Information Systems 2009 (IS'09)*, pages 135–142. IADIS Press, 2009.
- [URAS09] S. Ur Rahman, N. Aoumeur, and G. Saake. An Adaptive ECA-Centric Architecture for Agile Service-Based Business Processes with Compliant Aspectual .NET Environment. In *Proc. of ACM The 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008)*, pages 240–247. ACM Press, 2009.



## Appendix C

# N. Aoumeur Further Postdoctoral Publications

- [ABS05] N. Aoumeur, K. Barkaoui, and G. Saake. Coordination and Co-Nets for Specifying and Reconfiguring Agile information systems. In A. Van Aalst, editor, *2nd International Workshop on Applications of Petri nets to Coordination and Business Process Management, Join with Petri Nets Conference*, pages 1–16, 2005.
- [AFS03] N. Aoumeur, J. Fiadeiro, and G. Saake. Coordination Contracts Conceptualization and Validation Using Component-based Petri Nets. In *Proc. of the FM'03 Workshop on Formal Aspects of Components Systems (FACS'03), Pisa, Italy*, 2003.
- [Aou08] N. Aoumeur. Stepwise Rigorous Development of Distributed Agile Information Systems: From UML-Diagrams to Component-Based Petri Nets. *Enterprise Information Systems (Taylor and Francis Group)*, 2(2):121–156, 2008.
- [AS03] N. Aoumeur and G. Saake. Stepwise and Rigorous Development of Evolving Concurrent Information Systems : From Semi-formal Objects to Sound Evolving Components. In *In Proc. of the OOIS'03*, volume 2817 of *Lecture Notes in Computer Science*, pages 60–70, 2003.
- [AS07a] N. Aoumeur and G. Saake. Dynamic Interaction of Information Systems: Weaving Connectors on Component Petri Nets. In J. Cardoso and J. Cordeiro and J. Filipe, editor, *9th International Conference on Enterprise Information Systems (ICEIS'07)*, pages 152–158. INSTICC, 2007.
- [AS07b] N. Aoumeur and G. Saake. Features Interaction in Adaptive Service-driven Environments: A Reflective Petri Nets-Based Approach. In C. Rolland, O. Pastor, and J. Cavarero, editors, *Proc. of the First International Conference on Research Challenges in Information Science (RCIS 2007)*, pages 297–308. IEEE CS, 2007.

- [AS07c] N. Aoumeur and G. Saake. UML-driven Information Systems and their Formal Integration Validation and Distribution. In J.C. Augusto and J.Barjis and U. Nitsche, editor, *The 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS-2007—Workshop@ICEIS'07)*, pages 63–73. INSTICC, 2007.
- [AS08a] N. Aoumeur and G. Saake. A UML-Rewriting driven Architectural Proposal for Developing Adaptive Concurrent Information Systems. In *In Proc. of 7th International Conference on Information Systems Technology and its Applications (ISTA'08 @UNISCON)*, pages 393–404. LNBIP, Volume 5, 2008.
- [AS08b] N. Aoumeur and G. Saake. Modelling and Certifying Concurrent Systems: a MAUDE-TLA Driven Architectural Approach. In *In Proc. of of 5th International Conference on Information Technology : New Generations (ITNG'08)*. IEEE CS, 2008.
- [ASB07] N. Aoumeur, G. Saake, and K. Barkaoui. Incremental Specification Validation and Runtime Adaptivity of Distributed Component Information systems. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 123–136. IEEE Computer Society, 2007.