Goal-driven service mediation solution

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek, Informatie- en Communicatietechnologie
Colosseum 27, NL-7521 PV Enschede
The Netherlands

by:
Camlon H. Asuncion

Thesis for a degree in Master of Science in Business Information Technology
University of Twente, Enschede, The Netherlands

Graduation committee:

Internal:
dr. ir. Marten J. van Sinderen (UT- EEMCS)
dr. Maria-Eugenia Iacob (UT-MB)

External:
Menno Holtkamp, MSc (TNO-ICT)
dr. ir. Wout Hofman (TNO-ICT)

Enschede
The Netherlands
2009

Abstract

This thesis proposes a solution to the interoperability problem of enterprise systems in the context of service mediation. Service mediation is ideal when two or more systems want to interoperate but their functionalities, exposed as services, are fixed and are difficult, if not impossible to change, for the purpose of integration. A Mediator is a software that sits between two or more systems to resolve process and data mismatches. Process mismatches occur when collaborating systems use services that follow different ordering of message exchanges. Data mismatches occur when collaborating systems use different information models (or vocabularies) to describe the messages that are exchanged by their services

However, we find that designing integration solutions has always been traditionally technology-driven where Information Technology (IT) specialists do most of the job of building the technical solutions. We argue that business domain experts should be given the opportunity to participate more actively in the design of the mediation solution. We believe that their involvement is crucial in ensuring that the mediation solution delivers its intended business purpose.

This thesis investigates the use of goal-driven approaches to enable business domain experts in specifying and elaborating the requirements of the mediation solution. Goals capture the reasons why the integration is needed. Goals also provide business domain experts a sufficient level of abstraction in specifying and validating integration design choices at the business level and for communicating such choices among different stakeholders. As such, we argue that business domain experts are at the best position to describe the requirements of the service mediation solution through goals.

Finally, we investigate the use of model-driven techniques to transform these abstract goals into technology-specific implementations, and investigate how business rules can be used in the process. Model-driven development allows us to raise the problem and solution analyses spaces to a level of abstraction that is technology independent more suited for business-level analysis. We believe that model-driven techniques should give added flexibility to our solution as business requirements, their design specifications, and technical implementations are treated as separate concerns with the resulting artifacts maintainable and reusable.

Acknowledgements

This thesis culminates my two-year Master's program in Business Information Technology at the University of Twente, Enschede, The Netherlands.

I would not have been able to successfully complete this thesis without the patient tutelage and support of my supervisors during these past eight months: My thanks go to my supervisors from the University of Twente – Marten van Sinderen and Maria-Eugenia Iacob. It was because of Marten that I came to know of service mediation, and Maria who introduced me to her work on rule-based design of SOA systems. The contents, structure, and overall result of this thesis would have been realized if it were not for their valuable inputs. I also owe this thesis to my supervisors at TNO-ICT – Menno Holtkamp and Wout Hofman both of whom have given me what I considered to be an excellent dynamic working environment at TNO. Their advice on the content and structure on the thesis were also valuable. I also would like to thank other folks at TNO Enschede for always making me feel welcome: Frederik, Michael, Wout, Roel, Jasper, Dennis, Erwin, Matthijs, and Catalijna.

My thanks also go to Dick Quartel and Stanislav Pokraev of Novay whom I had the great pleasure of working with to solve the Payment Problem Scenario of the SWS Challenge. I thank Stan for his valuable inputs in structuring the first parts of my thesis, for introducing me to MDSL, and for giving me the opportunity to exercise my programming passion in Java. I thank Dick for assisting me in many ways especially during the simulation of the Mediator ISDL and for helping me understand the concepts behind ARMOR.

I would also like to thank the Royal Dutch/Shell Centenary Scholarship Fund for giving me the opportunity to come to the Netherlands in the first place. I also thank the folks at the International Office of the University of Twente for also assisting me in coming to Holland.

My thanks also go to a fellow BITer, Chris Jager, who has generously given his time to provide comments to my thesis.

Nagpapasalamat din ako ng lubos kay Jesette Campomanes, ang aking minamahal na kasintahan. Kung hindi din sa kanya, hindi ko maisip kung paano ko malalampasan ang lahat ng hirap sa paggawa ng aking thesis. Natulungan niya ako hindi lamang dito kung hindi sa maraming pang hindi ko na mabilang na paraan. Maraming salamat sa kanyang pag-unawa at pagsuporta sa aking pangarap na makapunta dito sa Holland upang makapag-aral. Maraming salamat din sa kanyang pamilya sa kanilang pagtulong sa akin habang ako'y wala sa Pilipinas.

Labaw sa tanan, gapasalamat ko ug daku sa akong mga pinaggang ginikinan, Raul ug Shirley Asuncion, ug igsuon, Shella Asuncion, sa ilang kanunay nga pag-ampo para sa akong malampuson nga pag-iskwela diri sa Holland ug sa ilang kanunay nga pagsuporta sa akong mga pangandoy. Para kini sa ila. Hinaot unta muabot ang panahon, malooy ang Ginoo, nga makabalos ko sa ilang kaayo.

Camlon Asuncion

September 21, 2009

Enschede, The Netherlands

Table of Contents

Figures

Tables

Listings

Abbreviations

ACE	Attempto Controlled English
ADS	Accounting Department System
AJAX	Asynchronous Javascript And XML
APE	Atttempto Parsing Engine
ARMOR	Architectural Modeling of Requirements
ASM	Abstract State Machines
BMM	Business Motivation Model
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
CBPL	Common Business Patterns Language
CIM	Computation Independent Model
CNL	Controlled Natural Language
CLIPS	C Language Integrated Production System
COSMO	COnceptual Service MOdelling
DERI	Digital Enterprise Research Institute
DRS	Discourse Representation Structure
DRT	Discourse Representation Theory
DSL	Domain Specific Language
EA	Enterprise Architectures
ECA	Event-Condition-Action
EMF	Modeling Framework Project
ERD	Entity-Relationship Diagrams
FIP	Financial Information Provider
GEF	Graphical Editing Framework
GORE	Goal Oriented Requirements Engineering
G&BR	Goals and Business Rules
IFIP	International Federation of Information Processing
ISDL	Interaction Systems Design Language
IT	Information Technology
jABC	Java Application Building Center
JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML Web Services
Jess	Java Expert System Shell
JessDE	Jess Development Environment
KAOS	Knowledge Acquisition in autOmated Specification
LHS	Left-hand side
LISP	LISt Processing

LPC	Lightweight Process Coordination
MDA	Model Driven Architecture
MDS	Management Department System
MDSL	Mapping Domain Specific Language
OMG	Object Management Group
PIM	Platform Independent Models
PO	Purchase Order
PRR	Production Rule Representation
PSM	Platform Specific Models
QoS	Quality of Service
RAM	Random Access Memory
RDF	Resource Description Framework
RE	Requirements Engineering
RFC	Request for Comment
RHS	Right-hand side
RIF	Rule Interchange Format
ROI	Return of Investment
RuleML	Rule Markup Language
SBVR	Semantics of Business Vocabulary and Business Rules
SD	Strategic Dependency
SD	Strategic Rationale
SEW	Service Engineering Workbench
SIB	Service Independent Building Blocks
SLG	Service Logic Graph
SOA	Services Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SWEET	Semantic WEb Enabling Technology
SWOT	Strength, Weakness, Opportunity, and Threat
SWRL	Semantic Web Rule Language
SWSC	Semantic Web Service Challenge
TNO-ICT	Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek, Informatie- en Communicatietechnologie
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
UMM	UN/CEFACT's Modeling Methodology
URL	Universal Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
WSML	Web Service Modeling Language
WSMX	Web Service Execution Environment
WSMT	Web Service Modeling Toolkit
WTP	Web Tools Platform
XSD	XML Schema Definition
XSLT	EXtensible Stylesheet Language Transformations
XML	Extensible Markup Language

1

Introduction

This thesis proposes a solution for service-oriented mediation of applications. In particular, we seek to provide non-technical business domain experts an active participation in the design of the solution enabled by goal-driven and model-driven approaches. This chapter presents the motivation, objectives, approach and the overall structure of this thesis.

The structure is as follows: Section 1.1 positions the reasons why this research was done. Section 1.2 describes research objective and questions. Section 1.3 presents the research approach that we adopt for achieving the research objectives and answering the research questions. Finally, Section 1.4 gives an overview of the general structure of this report.

1.1. Motivation

In today's era where there are strong demands to quickly adapt to rapidly changing business environments, no enterprise can ever afford to be an island. To a large extent, the competitiveness of an enterprise depends on its capabilities to foster innovativeness with other enterprises (Charalabidis, Gionis, Hermann and Martinez, 2008). With current advances in networking and computing technologies, autonomous enterprises are joining together to form *networks of enterprises* (often called *virtual enterprises*) so as to collectively achieve added value in their services and ensue exploration of more business opportunities (Van Sinderen, 2008).

Achieving such networks of enterprises, however, is not an easy task. Among the current challenges in enterprise computing, as described by Van Sinderen (2008), is providing *interoperability* solutions to the seamless collaboration of networked enterprises. Most of today's industries have investments in large legacy systems that were not originally designed to be interoperable, aggravated further by the oversupply of standards (or the lack thereof), proprietary developments or extensions, and heterogeneous hardware and software platforms. It is no surprise therefore that much research interest is currently being conducted in the area of enterprise interoperability to ensure growth and competitiveness of enterprises. Such example of this is one initiated by the International Federation of Information Processing[1] (IFIP). Another challenge is for enterprises to be *flexible* so that they can continuously adapt to current and future changes and demands from within and outside their business environment. This is especially true

[1] http://www.ifip.or.at/homeintro.html

when business goals need to be changed, temporary partnerships need to be created, technological changes need to be considered, etc. It is therefore imperative that such networked enterprises are designed for change so that they can quickly respond to emerging business opportunities (Van Sinderen, 2008).

This thesis responds to these challenges, *firstly*, and in particular, by providing a solution to mismatching process specifications of interoperating enterprise systems through *service mediation*. Service mediation is ideal when systems need to interoperate but have existing and often difficult-to-change services. Matching the sequence of message invocations between such services is oftentimes difficult, if not impossible, as these systems may follow proprietary or standards-based message structures. A Mediator therefore is a piece of software that acts as an intermediary in reconciling message protocol and semantic data mismatches (Quartel, Pokraev, Mantovaneli Pessoa and Van Sinderen, 2008a).

Secondly, we want to provide flexibility to our mediation solution by keeping it business-driven; that is, we want to keep the solution problem-oriented rather than technology-oriented. A change in the business requirements should not adversely affect the underlying technology implementation. We keep it business-driven by providing direct involvement of *business domain experts* during the integration design. The solution should allow them to see how their business requirements are aligned with technological implementations.

However, designing integration solutions has always been traditionally technology-driven where Information Technology (IT) specialists do most of the job of building solutions based on such technologies as WSDL, BPEL, etc. Business domain experts are merely consulted at the early stage of requirements elicitation. Even if business domain experts do get involved in the collaboration design, they will have to contend with learning such technologies and sophisticated tools which may be too technical for them. (Pokraev, Quartel, Steen, Wombacher and Reichert, 2007; Hofman, 2008a). Furthermore, infusing the participation of business domain experts in integrating service-based systems is often difficult because technology standards (e.g. WSDL) are inherently defined with so little business semantics that business people do not understand them. For business people to match and compose services at the technology level is a daunting task. It is difficult therefore to compose the integration solution using these standards at the business level (An and Jeng, 2007).

In our objective to keep our mediation solution business-driven, we investigate whether a *goal-driven approach* can be used by business domain experts in specifying the requirements specifications of the mediation solution. Goals are high-level objectives of a business, organization, or system. They capture the reasons *why* a system is needed and guide decisions at various levels within the enterprise (Anton, McCracken and Potts, 1994). Goals are important as they can be used to elaborate system requirements. They also provide decision makers a sufficient level of abstraction in specifying and validating system design choices at the business level, and for communicating such choices among different stakeholders (Iacob, Rothengatter and Van Hillegersberg, 2009). As such, we argue that business domain experts are at the best position to describe the requirements of the service mediation solution through goals.

Once we have the requirements of the service mediation solution expressed in terms of goals driven by business requirements, we further investigate the use of *model-driven techniques* to transform these abstract goals into technology-specific implementations, and investigate how business rules can be used in the process. Model-driven development allows us to raise the problem and solution analyses spaces to a level of abstraction that is technology independent more suited for business-level analysis (Miller and Mukerji, 2003). We believe that model-driven techniques should be able to give added flexibility to our solution as it treats business requirements, their design specifications, and technical implementations as separate concerns with the resulting artifacts maintainable and reusable.

Realizing the need for a business-level design of services, the *Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek*, division of *Informatie- en Communicatietechnologie* (TNO-ICT), is currently working on a project, called the *Service Engineering Workbench (SEW),* which aims to

provide a tooling environment that allows technology-independent, business-viewpoint design and choreography of services. Information (semantic) and behavior (process) requirements of business transactions, interactions, and protocols between service consumers and providers serve as inputs to the SEW. These requirements are modeled at the business level which can later be transformed into technology specifications and implementations which serve as its outputs (Hofman, 2008b). As the project is still at its early stages, we aim to contribute to this workbench by exploring service mediation as one of its solution components. The mediation solution will detail requirements that can be supported by the SEW. Consistent with SEW's general requirements, we strive to provide technology-independent, business-level design of services and their transformation to platform specific implementations in the context of service mediation.

1.2. Research objective

Our main research objective in *general* is:

> **To improve the participation of business domain experts in the design of networked enterprises**

We *hypothesize* that our general objective can be achieved by:

> **Adopting a service-oriented mediation approach and applying goal-driven and model-driven techniques in the design**

Specifically, the following research questions (*RQ*) have to be answered:

RQ1: Who are the business domain experts? Why and in what ways should business domain experts be involved in the design of a service mediation solution?

RQ2: What are goals? How are they used to specify software requirements? What goal-driven approaches are there? How can goal-driven approaches be used to specify requirements for the design of the mediation solution?

RQ3: What is service mediation? What service mediation approaches are currently available? How do they involve business domain experts in the design of an integration solution?

RQ4: How can goal-driven and service mediation approaches be combined to allow business-level design of the service mediation solution? How can model-driven techniques be used to combine these approaches?

RQ5: What available tools are there to help in the (semi-)automated Mediator design, goal modeling, and model transformations?

RQ6: What requirements can be drawn from the service mediation solution which can serve as inputs to the design of the Service Engineering Workbench of TNO-ICT?

1.3. Research approach

The development of this research shall follow research and design methodology principles proposed by Wieringa (2007; 2008). We show graphically our research approach in Figure 1.

As this research attempts to devise a way to improve the involvement of business domain experts in the design of networked enterprises, we thus view our research as largely based on solving a *practical* problem; that is, we want to reduce the difference in the current and desired state of the world. In particular, we tackle *selection* problems where we seek one solution from out of a known set, and the ensuing *implementation* problems where we determine ways to best realize the

selected solution. To some extent, this research also deals with solving *knowledge* problems; that is, we want to reduce the gap between current and desired knowledge states. Following the engineering lifecycle of Wieringa (2008), we divide our research methodology into three phases: *problem analysis*, *solution design*, and *solution validation*.

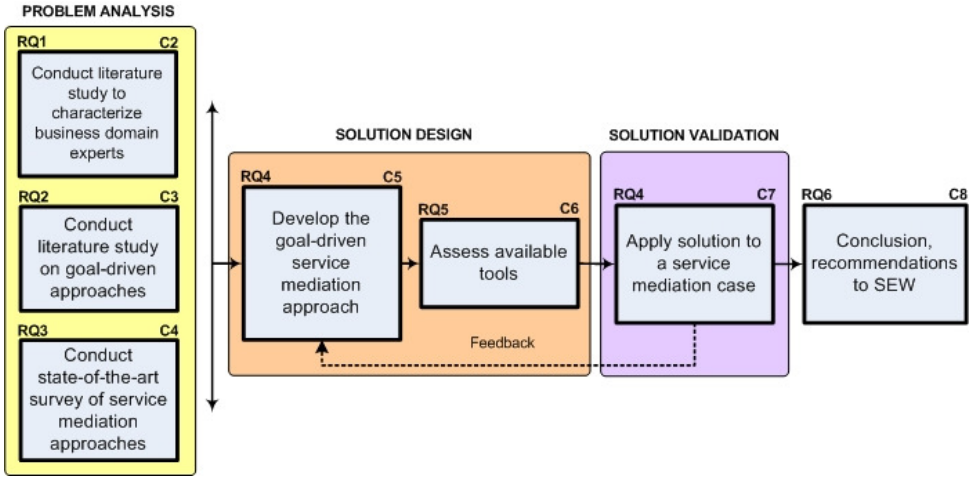


Figure 1-1: Research approach

During *problem analysis*, we solve both knowledge and selection problems. We first do a literature study to describe who business domain expert are and the necessity of their involvement during system design – this is largely a knowledge problem. We then do another literature study of goal concepts, goal-oriented requirements engineering and modeling concepts, and how goals can be implemented as business rules (a knowledge problem). This literature study includes a survey of goal modeling approaches to see how they can specify requirements for the design of the mediation solution; that is, we look into how they use goals as basis for specifying the *Enterprise Architecture*(EA) design of the Mediator (a selection problem). We then do a state-of-the-art literature survey to determine and select an appropriate service-oriented mediation approach (a selection problem). In particular, we investigate how these approaches involve business domain experts in the design of their mediation solution. In addition to these literature studies, our selection will also be motivated by looking into current established research work in similar areas within our community. We answer research questions 1, 2 and 3 in the process, and report the results in Chapters 2, 3 and 4, respectively.

During *solution design*, we solve an implementation problem. We determine the best way to combine the selected goal modeling and service-oriented approaches that allows better involvement of business domain experts during the design process. We also describe which tools are available in the industry and academe that provide partial or full automation of mediation design, goal modeling and transformations. We answer research questions 4 and 5 in the process, and report the results in Chapter 5 and 6, respectively.

Lastly during *solution validation*, we apply our solution to two case studies for illustration and validation purposes. We assess how the solution allows better participation of business domain experts in the design through case study using the Semantic Web Service Challenge (SWS) Payment Problem Scenario[2]. During the validation process, we use new knowledge we can gain as feedback to the solution design phase. We answer research question 4 in the process, and report the results in Chapter 7.

Finally, we use the results of this research to propose design considerations to the SEW. We answer research question 6 in the process, and report the results in Chapter 8 along with the conclusion, other recommendation, limitations, contributions, and possible future work for the thesis.

[2] http://sws-challenge.org/wiki/index.php/Scenario:_Payment_Problem

1.4. Report structure

Congruent to the research objectives, questions and approach in Figure 1-1, we structure the report into three parts: problem analysis (Part I), solution design (Part II), and solution validation (Part III). Please refer to Figure 1 to see how the chapters map to the research questions.

Part 1: *Problem Analysis*

C2: **Chapter 2** reports the result of a literature study that describes who business domain experts are. We also describe why their involvement during the design is necessary and the ways in which they can manifest their involvement.

C3: **Chapter 3** reports the result of a literature study on the concepts of goals, their classification, and activities related to goal oriented requirements engineering. We also show the relationship between goals and business rules, provide a basic classification of the latter relevant to this research, motivate their importance, and provide recommendations when to use them in the design of software systems. This chapter concludes with a survey of goal modeling approaches aimed at observing how they can specify requirements for the design of the mediation solution; that is, we look into how they use goals as basis for specifying the *Enterprise Architecture* design of the Mediator (a selection problem).

C4: **Chapter 4** presents the results of a literature study that clarifies the concept of service mediation, including the assumptions we applied to this thesis. It also presents the results of a state-of-the-art literature survey of service mediation approaches that compares them according to how they involve business domain experts in the design of integration solutions.

Part II: *Solution Design*

C5: **Chapter 5** presents our goal-driven service mediation solution in detail. The *solution* is comprised of both a *methodology* and *architecture*. We first describe the three approaches we have used as basis for our solution which includes a *goal-driven approach* for requirements engineering in Enterprise Architectures, a methodology for *service mediation,* and a framework for *model-driven design* of service systems using goals and business rules. We then provide an overview of the technologies we used to realize each step of the methodology and architecture. Specifically, we show architecturally how business domain experts can have goals of the integration, expressed in terms of refined business rules, be formulated, validated and transformed into design and executable architectural artifacts. We strive to keep the methodology as generic as possible so that other technologies may be used in lieu of those we have chosen to realize our solution. This essentially keeps our methodology problem-oriented rather than technology-oriented.

C6: **Chapter 6** presents available commercial, open source, and academic software tools we used to carry out the design and implementation of the methodology and architecture.

Part III: *Solution Validation*

C7: **Chapter 7** reports how our solution is applied through a case scenario. We take the Semantic Web Service (SWS) Challenge as an illustrative and validating case.

C8: **Chapter 8** summarizes this report by providing conclusions, recommendations to the SEW, contributions, limitations and future work.

Part I

Problem Analysis

This part is comprised of Chapters 1, 2, and 3. The objective of this part is to report the results of a literature study on the concepts of business-level design and roles and skills of business domain experts in the design of integration solutions. Next to this a literature study on the concepts of goals, goal oriented requirements engineering, and current goal modeling approaches is undertaken. Also a literature study on the concepts of service mediation is performed. Finally a state-of-the-art literature survey on service-oriented mediation approaches is included.

2

The business domain expert

This chapter introduces who the business domain experts are as used in this research, why their involvement during the design of software systems is necessary, and the ways in which they can manifest their involvement.

The structure is as follows: Section 2.1 describes our understanding of the business domain expert. Section 2.2 describes the importance of their involvement in the design of integration solutions. Section 2.3 describes ways in which they can be involved. We end this chapter with a summary in Section 2.4.

2.1. Who is the business domain expert?

This section describes how we define who *business domain experts* are in the design of software systems. To do this, we first review the literature to see how they are characterized in various research areas. We formalize our definition at the end of this section.

Findings

In the field of Requirements Engineering (RE) research, business domain experts are *stakeholders* – "an individual or group who is actively involved in the project, who is affected by the project, or who can influence its outcome" (Wiegers, 2006). They are those who gain or lose something (e.g. functionality, revenue, status, compliance with rules, etc.) as a result of the development of the software system. More specifically in this research, they are those whose skills are needed to build the system (Alexander and Robertson, 2004). These stakeholders may have competing interests that cause the acquisition, specification, and evolution of requirements to be inconsistent (Ryan and Greenspan, 1996; Alexander, 2007).

In the area of service mediation research, business domain experts do not necessarily "speak" the language of IT experts. They do not have or want to have the technical knowledge to understand how these requirements are implemented at the IT level (Quartel, Pokraev, Pessoa and Van Sinderen, 2008a). They have detailed knowledge about how the business logic of a process that is to be designed, but are not computer programmers and may not even have a technical background (Steffen, Margaria, Nagel, Jorges and Kubczak, 2006).

In the context of business-level service composition, business domain experts are able to describe system requirements from the business perspective. However, they are not equipped to

implement these requirements into technical implementation such as specifying them in service description (e.g. through WSDL, BPEL, XSD) because of a lack of background in these technologies. Furthermore, they may not have the faculty to select, from among an array of existing or required services, the right service to form a composition (Ying and Li, 2007).

In the design of Model Driven Architectures (MDA), business domain experts have expertise in the business domain and can provide the requirements of the system as imposed by the business. They do not need to know how these requirements are realized as functionalities of the systems implementing these requirements. In contrast, IT experts are those who have the technical expertise to design and construct the business requirements as software artifacts (e.g. architecture, model, code, technical documentation, etc.) that satisfy those requirements (Miller and Mukerji, 2003)

In the area of business-driven service personalization, business domain experts are able to demonstrate a mastery of their own business; however, they are only able to understand the functionalities of an IT system when these functionalities are expressed in business terms (also called *Business Semantics*). Their interests may be different from other business domain experts and may only wish to know concrete business details on areas they are interested in and only basic information in others (also called *Diversity of Interests in Business Functions*). Next to this, they may have different, and often conflicting, demands over what functional or non-functional properties a system should have, even if their requirements are similar at the higher level (also called *Personalized Business Requirements*) (Wang, Yu and Han, 2005).

Summary

Summarizing the salient characteristics, a business domain expert is:

- Is a stakeholder to the realization of the integration solution,
- Has the expertise or mastery of his respective business domain (e.g. finance, health, banking, insurance, etc.),
- Is able to describe the requirements of the system from the business perspective,
- May have conflicting interests with other business domain experts as to how the requirements of the system will be specified and eventually implemented, and
- Does not have (or does not *want* to have) technical background or expertise to understand how business requirements are translated into their technical implementations.

Applying these observations to our research, we thus define a **business domain expert** as:

> **A stakeholder who has the expertise of a business domain, and whose expertise is used as basis to specify the business requirements (which may also run in conflict with those of the other business experts in the same or different domain) of the integration solution. However, he does not necessarily have (or does not want to have) the technical knowledge to translate the business requirements into their technical implementations.**

Comparing business domain experts to other participants of the integration solution

We distinguish in this thesis a business domain expert from that of a *business analyst* and an *IT expert*. We use the term *business analyst* as described Vongsavanh and Campbell (2008) and Lister (2009) (seeing that there are several confusions as to their names as reported in literature). A business analyst is someone whose main *role* is to liaise between the business and IT professionals so as to elicit, decompose, validate requirements and assess related risks. They do not necessarily have the same business domain knowledge as those of the business domain experts or do not have the same technical expertise as with IT experts. Their functions and backgrounds may thus

operate from either the IT or business side of the organization. As such, a key *skill set* among business analysts is their ability to display fundamental consulting skills such as communication, listening, writing, presenting, abstraction and negotiation by keeping a close working relationship with the business community. This thesis views them as mediating between business domain experts and IT experts. This thesis assumes that business analysts are familiar with goal oriented requirements engineering activities and modeling concepts.

If business analysts are concerned with the business and how to use IT to achieve business goals, then *IT experts* are more concerned with software development and implementation. IT experts are regarded as having a technical *role* with more emphasis placed on business process and system design. As with business analysts, their *skill sets* include communication and technical, general analysis, design and problem solving ability (Vongsavanh and Campbell, 2008). This thesis uses the term IT expert to loosely refer to system architects, systems analysts, developers, programmers, quality assurance testers, etc. We further assume in this research that they are knowledgeable of service specifications (e.g. WSDL, BPEL, SOAP, etc), Services-Oriented Architecture (SOA) concepts, or service-oriented design principles (e.g. service coherence, reusability, granularity, autonomy, parsimony, etc.), Model Driven Architecture (MDA) concepts, and rule-based technologies.

To model their relationships, we use the *stakeholder map* (also known as the *onion model*) proposed by (Alexander and Robertson, 2004) that distinguishes their level of *involvement* in the design of integration solution: the closer they are to the innermost center circle, the greater is their involvement. The model is shown in Figure 2-1.

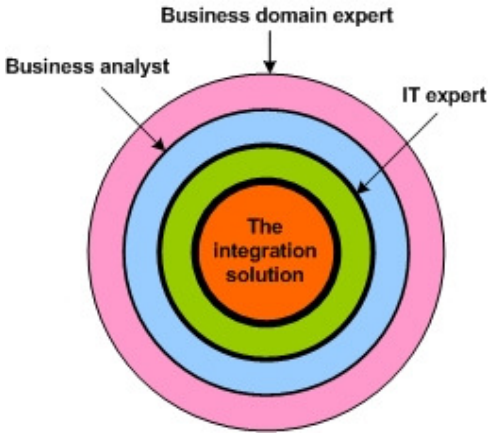


Figure 2-1: The integration solution stakeholder onion model

2.2. Why involve them?

Requirements initially come from preliminary materials given to the requirements engineer. More often than not, however, most of the requirements come from stakeholders (Van Lamsweerde, 2001). Alexander and Robertson (2004) argue that maintaining the involvement of stakeholders in the design of software systems is crucial to successfully implementing software projects. The consultant Tim Lister[3] regards project success as "meeting the set of all requirements and constraints held as expectations by key stakeholders" (Wiegers, 2006).

However, studies show that inadequate involvement of stakeholders is one of the leading causes of software projects failure. The Standish Group research in 2001 reports that of the 280.000 projects that were studied, 137.000 projects were challenged (49%), while 65.000 (23%) failed entirely. Project failure was not caused by the lack of money to sustain the project but, to some

[3] http://www.systemsguild.com/GuildSite/TRL/Tim_Lister.html

extent, the lack of user involvement (second to lack of executive support). A project can still fail even if it is delivered on time and within the allotted sources if the needs of the users were not properly addressed.

User involvement in the design of information systems has been a much researched area. And the general conclusion has been that their involvement is critical to the successful design of software applications, especially in large-scale projects. In fact, it is considered good practice as evidenced by the wide array of available methodologies available today (e.g. prototyping, agile programming, etc.). Involving users ensures their commitment, reduces resistance to change, increases satisfaction and acceptance of the system. Conversely, the lack of user involvement ensue high failure rates in terms of the software's implementation and use, late delivery of the software, over-budget implementations, scope creep, user backlash, and limited technical functionality, among others (Wagner and Piccoli, 2007).

Particularly in this research, the importance of the business domain expert in the design of integration solution can be appreciated more when several enterprises wish to have their enterprise collaborate at a massive scale. The complexity involved in this exercise is no easy task; and it is hard to imagine if all the integration design is done by IT experts lest we fall to the same old requirements problem where requirements are misidentified, inaccurate, or insufficient. Furthermore, collaborating enterprises may not belong to the same business domain; for example, a large hospital may want to perform a collaborative business venture with an insurance company through the interoperation of their enterprise systems. Business domain experts from these two fields have expertise from entirely different domains. Their expertise is therefore needed to accurately and completely specify and reconcile integration requirements.

In conclusion, getting the right software requirements is therefore critical to getting the right software (Van Lamsweerde, 2008). And we argue that business domain experts play a vital role in requirements specification as they drive and guide the requirements elicitation process. Their role is all the more important for the successful implementation of integration solutions between large, mission-critical, enterprise systems.

2.3. How can they be involved?

This thesis looks at the following ways in which business domain experts can be involved in the in the design of integration solutions:

By serving as the source of the requirements

Stakeholders have an interest in the requirements since they have an interest in, or an effect on, the outcome of the product. They have an interest the product because they want it to do their work correctly. Moreover, they want their work to be successfully implemented. The importance attached to stakeholders comes from the fact that they are the source of the requirements (Robertson, & Robertson, 2006).

By validating the requirements specifications

To ensure the quality of the integration solution, business domain experts can validate requirements specification. Requirements validation ensures quality by detection of defects of the requirements, reporting them, analyzing their causes and doing the necessary corrective actions to fix them. The earlier the error in the requirements is detected, the less costly it is to repair it. To validate requirements, business domain experts can perform *inspection and reviews* to analyze the defects requirements specifications (Van Lamsweerde, 2009, p.187).

By validating solution artifacts

In business process modeling, even when workflow models are syntactically or structurally correct (e.g. free from deadlock occurrences), it does not necessarily mean that such models have captured the specified business logic. The knowledge of business domain experts is thus of paramount importance to determine if the business logic, business rules, and other requirements have been met (Sadiq, Orlowska, Sadiq and Foulger, 2004). Thus, during integration design, business domain experts can validate the business logic of the integration's business process model.

By providing semantic information to the mapping of data elements

When collaborating systems interoperate through their existing service operations, message-level heterogeneities may exist. For example, one system may not have the same semantic interpretation of the message data passed by the other (semantic heterogeneity). On the other hand, naming conflicts may arise where the meaning of a message data is interpreted the same by both services only that the message is structured differently (syntactic heterogeneity) (Nagarajan, Verma, Sheth and Miller, 2007). During the integration design, business domain experts from collaborating enterprise can come together to resolve these message heterogeneities.

By specifying modifications to the requirements

Modifications to the integration requirements may come in a form of maintenance activities. Several types of maintenance such as corrective, adaptive and perfective maintenance best suit the involvement of the business domain expert. In the *corrective* maintenance, the business domain expert needs to identify the things that are not functioning properly. These therefore were referred to as the bugs, misinterpreted designs, missed specified designs, and ignored needs. The *adaptive* maintenance takes place as to meet the changing demands of the environment such as government regulations and improving old policies and procedures. Involvement in the *perfective* maintenance counts on improving the existing system to upgrade and efficiently perform the routine functions (Alge, and Upright, 2009).

2.4. Chapter summary

This thesis defines a business domain expert as a stakeholder to the integration process. He belongs to some business domain (e.g. banking, health, insurance) of an enterprise. He manifests his expertise by his ability to specify the requirements of the integration solution. However, he does not necessarily have (or does not want to have) the technical knowledge to translate the business requirements into their technical implementations.

To assist business domain experts during the integration design, a business analyst may liaise in his behalf with the IT expert. A business analyst does not necessarily have the business expertise as the business domain expert but has knowledge of eliciting, decomposing, validating requirements of the integration; furthermore, they may not have the same technical expertise as IT experts. We loosely coin the term IT expert to refer to an individual who has the ability to realize the requirements into their technical implementations.

Involving the business domain experts during the design is crucial to successfully implementing the integration solution. Their involvement ensures their commitment to the integration project, reduces resistance to change, increases satisfaction and acceptance of the system. Conversely, the lack of user involvement ensue high failure rates in terms of the software's implementation and use, late delivery of the software, over-budget implementations, scope creep, user backlash, and limited technical functionality, among others. Their involvement is especially critical when

several enterprises wish to have their enterprise collaborate at a massive scale, and when enterprise systems belong to different business domains.

Business domain experts can participate during the integration design (i) by serving as the source of the requirements, (ii) validating the requirements specifications, (iii) by validating solution artifacts, (iv) by providing semantic information to the mapping of data elements, and (v) by specifying modifications to the requirements.

3

On goal-oriented system design

This chapter looks into the concept of goals and how they are used to specify software requirements. We also show how goals can be implemented as business rules for the design of software systems. Finally, we show what goal-driven approaches are currently available, and compare them as to how they can be used to specify the requirements for the design of the service mediation solution.

The structure is as follows: Section 3.1 describes the definition of goals as used in this thesis. Section 3.2 provides a general classification of goals. Section 3.3 describes the importance of goals in requirements engineering. Section 3.4 describes activities related to Goal Oriented Requirements Engineering (GORE). Section 3.5 describes goals and their relation to business rules including a basic definition and classification of rules. Section 3.6 provides an overview of the goal-driven approaches reported in literature. Section 3.7 provides a summary of this chapter.

3.1. Definition

Today, developing the right software largely depends on getting the right software requirements. This is the study of Requirements Engineering (RE) which deals with "the elicitation, evaluation, specification, consolidation, and evolution of the objectives, functionalities, qualities, and constraints that a software based system should meet within some organizational or physical setting". RE, however, has always been a very difficult task intrinsically. But, many consider that it bears a critical impact on the quality of software that can be produced. Poor requirements specification is widely believed to cause frequent, persistent, expensive and recurrent types of software errors leading to serious project cost overruns, delivery delays, failure to meet customer expectation, etc. (Van Lamsweerde, 2008).

Faced by the inadequacy of traditional approaches to RE, especially in dealing with complex software systems, researchers, since the 1990s, have looked at how objectives that a system should achieve or provide can best be captured from various levels of abstraction. They started to treat these system objectives or goals as first-class citizens in the RE world. This is in contrast with early practices where requirements focus only on processes and data without capturing the rationale of the software systems, making it difficult to relate requirements to the higher problem domain. Furthermore, traditional RE approaches tend to focus only on the software-to-be without considering the environment and assumptions that the software will have to eventually interact with (Lapouchnian, 2005). This gave way to the concept of *Goal-Oriented Requirements*

Engineering (GORE) which investigates how goals can be used for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying software requirements (Van Lamsweerde, 2001).

A widely accepted definition of **goals** is given by Anton, McCracken, and Potts (1996):

> **"Goals are high-level objectives of the business organization or system. They capture the reasons why a system is needed and guide decisions at various levels within the enterprise."**

Van Lamsweerde (2009) adds to this definition by stating that:

> **"Goals are prescriptive statements of intent that the system should satisfy through the cooperation of its agents."**

Goals are *optative* properties of a system under development; that is, they express *intentions*, *wishes*, or *desires* and are thus declarative. This is in contrast with *descriptive* statements which describe real facts (Jackson, 1995). For example, "*Ascend to the third floor*" is a goal stated declaratively, while "*The third floor is empty*" is a descriptive statement.

Agents are required to cooperate with one another by playing a *role* to satisfy and achieve a goal. They can be either be a human being (e.g. organizational actors, operators, end users, etc.), devices (e.g. sensors, actuators, etc.), existing software components (e.g. legacy, COTS, etc.), or new software components. They perform such roles in a way that restricts the behavior of a *system* (i.e. the system-to-be or the system-as-is). For example, the statement:

```
A student should first fully pay his tuition fee
before he can be allowed to graduate
```

List 3-1: A sample goal

This prescriptive statement is a goal which requires several agents to cooperate before a student is allowed to graduate. These agents may include for example the University's enrollment system, the student himself, the University's Admission Personnel, the Professor, etc. One can see that the finer the goal is, the less the number of agents required to achieve the goal.

Goals can be formulated at different levels of abstraction (also called *goal granularity*). Strategic or high-level goals can represent some generally desired situation (high granularity) such as "*improve company image*", "*attract more international students*", "*satisfy customer*", etc. They can also represent low-level technical concerns (low granularity) such as "*keep door closed while ascending or descending*" as in elevator systems (Van Laamswerde, 2009, p. 261).

3.2. Goals classification

Several authors propose various goal classification schemes. This research, however, only gives an overview of a relevant general classification of goals. We compare strategic vs. operational goals, functional vs. non-functional goals, and hard goals vs. soft goals. For an extensive classification, we refer you to the work of Van Lamsweerde (2001, 2009) specific to the KAOS approach for GORE.

Many authors, however, treat these classifications similarly. Some authors equate hard goals and soft goals to functional and non-functional goals, respectively (Dardenne, Van Lamsweerde and Fickas, 1993) and (Bresciani, Perini, Giorgini, Giunchiglia and Mylopoulos, 2004). Soffer and Wand (2005) argue that operational goals can be related to hard goals which serve as basis of functional goals, while strategic goals relate to soft-goals which serve as basis for non-functional goals. Furthermore, Jureta, Faulkner and Schobbens (2006) suggest that, while it is common to equate nonfunctional goals and soft goals, the latter should belong to another classification where they opposed to hard goals. This research makes such distinctions.

3.2.1. Strategic vs. operational goals

Soffer and Wand (2005), in their study of incorporating goals in business process modeling, propose a distinction between strategic goals and operational goals. A business process is a set of partially ordered activities aimed at reaching a goal. Strategic goals are more abstract objectives that organizations want to achieve. They cannot usually be performed by a specific process but nevertheless important for the organization.

```
To be the leading University in the Netherlands
```

List 3-2: A sample strategic goal

Operational goals define the desired state to be achieved which can be accomplished by a specific process.

```
Handle student enrollment
```

List 3-3: A sample operational goal

3.2.2. Functional vs. non-functional goals

Functional goals underlie services that a system is expected to deliver; that is they describe what the software does (Keller, Kahn and Panara, 1990). They state the intent behind a system service like satisfying an agent's request, informing the agent of the system's status, providing responses to specific events, etc., (Van Lamsweerde, 2009). In most cases, they can be assigned to a specific process, actor or organizational structure (Iacob, et, al. 2009).

```
Send purchase order invoice
```

List 3-4: A sample functional goal

Non-functional goals refer to expected system qualities such as security, safety, performance, usability, flexibility, customizability, interoperability, and so forth (Keller, et al., 1990). They refer to quality requirements that the software needs to satisfy while delivering the services (e.g., how the software provides security) (Jureta, et al., 2006). They do not apply to any specific process, actor or organizational structure. Both functional and non-functional goals can be both clearly defined and are thus measurable (Iacob, et al., 2009).

```
The customer should immediately receive payment receipt in one second
```

List 3-5: A sample non-functional goal

3.2.3. Soft vs. hard goals

Soft goals are goals whose satisfaction cannot be established in a clear-cut sense (such as for example in for modeling and analyzing non-functional, quality-related requirements) (Mylopoulos, Chung and Yu, 1999). Their criteria for satisfaction are typically abstract and may be subject to interpretation. Lacking objective criteria to measure achievement, they thus involve subjectivity where evaluating its achievement depends on its stakeholders (Jureta, et al., 2006). They specify high-level qualitative objectives that are difficult to measure (Iacob, et al., 2009).

```
The student's enrollment system should be secured
```

List 3-6: A sample soft goal

Hard goals are goals whose satisfaction can be established through verification techniques (Dardenne, et al., 1993; Darimont and Van Lamsweerde, 1996). Their criteria for satisfaction are clear and precise so that they can be measured quantitatively.

```
Increase product sales to 5%
```

List 3-7: A sample hard goal

3.3. Why use goals in engineering requirements?

The authors Yu and Mylopoulos (1998), Van Lamsweerde (2001) and Van Lamsweerde (2009, p. 272) enumerate the importance of goals in the engineering of requirements in the following manner:

- *Specification refinement.* Goals provide a natural way for structuring complex specifications. Such structure can be accomplished by defining a goal into a set of sub-goals that together contribute, satisfy or achieve it. Furthermore, goals can be used to clarify requirements especially in situations where requirements are difficult to make precise. For an instance, in describing non-functional requirements of a system-to-be such as flexibility, robustness, reusability, etc.

- *Requirement rationalization.* As a consequence to goals being refinable, we are able to reason the existence of a specific requirement especially in situations where its rationale may not be clear by simply looking at the goal's refinement. Goals and their refinements can also be used as basis to explain the requirements to various stakeholders. Requirements are often unclear when first elicited from clients and stakeholders.

- *Requirements identification.* Goals serve as the starting point for the identification of requirements that support such goals. This is especially important when goals appear in the identification and evaluation phases of requirements engineering process. Goal identification naturally involves repetitively asking "*why*", "*how*", and "*how else*" questions. Doing so allows us to explore various alternatives to achieve such goal, and may in fact give ways to new requirements that can contribute to it. This prevents stakeholders from prematurely adopting certain technological solutions right away.

- *Business-IT alignment.* Goals have been used as an important mechanism for connecting requirements to design seeing that requirements should ultimately lead to the design and implementation of the system-to-be. Goal refinements can be used as basis to describe to various stakeholders how the organization's strategic objectives, refined into goals, will be aligned to the system that will be developed, and back. Today, requirements no longer solely focus on what the system-to-be is supposed to do. Requirements should also be able to motivate how such systems will be able respond to the more dynamic nature of organizational environments.

- *Risk analysis.* Goals have been used to analyze potential risks which can result from the loss of achieving some goal. Goals can be used to create goal negations that prevent the goal from ever being satisfied. Goal negations allow possible exploration of countermeasures as new goals to reduce, mitigate, or prevent risks from occurring. This is especially helpful in critical safety- and security-critical systems.

- *Managing conflicts among requirements.* Goals can be used to manage conflicts among requirements. Even with a single set of requirements, possible conflicts may occur especially among non-functional requirements, such as, for example, tradeoffs that need to be made between cost and performance. Such conflicts may come from different viewpoints and concerns of various stakeholders. To manage conflicts at the goal level, conflicting goals must first be detected, various conflict-resolution goals explored, with the best option selected and propagated to the requirements level.

- *Structuring agent dependency*. Goals are eventually achieved by cooperating agents. Goals and their refinements can thus allow us to capture the agents that need to perform the goals and their relation with one another while distinguish their respective definitions, capabilities and responsibilities.
- *Delimiting scope*. The system to be developed is bounded according to how agents cooperate together to satisfy the goals of the system. These agents are both "good" and "bad" agents (with the former trying to perform the goal while the latter obstructing it). These agents define the scope of the system to be developed.
- *Reasoning about alternatives*. Goals can be refined through various alternative combinations of its sub-goals where the fine-grained goal can be operationalized through alternative system services and assigned to alternative agents. Negative goals can be avoided by considering alternative countermeasure goals. Conflicting goals can be managed by exploring alternative resolution goals. Goals are thus able to provide alternative options explicitly.
- *Requirements traceability*. When there are changes to the requirements, it is important that such changes are managed throughout the engineering process by ensuring that the impact of the change is localizable to manage and propagate. Goal refinements allow us to determine where a requirement comes from, why it came from there, and where it goes to next.
- *Evolution support*. A lower-level goal may be one particular way of achieving a higher-level goal. This lower-level goal may likely to change in the future as new ways may be identified to better achieve the higher-level goal. The higher the goal, the higher is its stability. This means that we can isolate higher-level goals which are stable from lower-level goals which may change in the future.

3.4. GORE activities

In GORE, the main purpose of goals is to aid in the elaboration or elicitation of requirements that support them. Goals are treated as high-level objectives of the organization or system. The elicitation of goals can be done in a top-down, bottom-up or hybrid fashion. Top-down approach usually starts from analyzing abstract strategic goals of an organization which are then refined into a set of soft goals refined further until they become hard goals. Bottom-up approach starts from analyzing individual actors and goals which are then aggregated into more abstract or higher-level goals that represent the desires of the organization (Iacob, et al., 2009).

GORE is *problem-oriented* RE which focuses on understanding requirements from a problem-oriented view (Wieringa, 2004). It takes a system-engineering approach to the modeling and analysis of the problem domain. The goal model describes the problematic phenomena, their relations, why they are seen as problematic and which stakeholders see them as so; they help reason about the purpose of the proposed solution, analyze and demonstrate which goals realize other goals including those that negatively contribute to their achievement (Quartel, Engelsman, Jonkers and Van Sinderen, 2009c).

3.4.1. Goal analysis/identification

Goal analysis, a top-down approach to goal elicitation, is the process of exploring gathered documentation for the purpose of identifying, organizing and classifying goals. The objective of this step is to build a preliminary draft of the goal model, and this is usually done at the early phases of requirements elicitation (Van Lamsweerde, 2009). This is a top-down approach to goal elicitation.

The information for goal analysis can come from high-level organizational or enterprise goals to system specific requirements. They may also come from strategic objectives of the organizations,

business goals and policies. Similarly, analyzing the current objectives and problems of the system-as-is (and their negations) could bring forth opportunities for identifying goals of the system-to-be. (Van Lamsweerde, 2009). Requirements may also come from stakeholders themselves so that they have to be extracted through interviews and documented. Other sources include process descriptions, flow charts, Entity-Relationship Diagrams (ER), preliminary documents, domain knowledge (Anton, 1996).

When requirements are given in a textual form, goals may also be extracted by *searching* for statements which seem to guide design decisions at various levels within a system or organization. Searching for action words is a useful way of extracting goals from stakeholders' description (Anton, 1996). These action words are intentional keywords which may be of the *prescriptive*, *intentional*, or *amelioration* types. Keyword searching is a simple, cheap, and quite an effective way of identifying preliminary goals. The preliminary goals are then temporarily given a name, a rough preliminary information definition, and assigned into some classification in terms of the goal taxonomies defined at the meta-level. Examples of keywords that can be used for goal identification are given below (Van Lamsweerde, 2009, p. 310):

```
shall, should, must, has to, to be, may never, may not, should never…
```

List 3-8: Prescriptive search keywords for goal identification

```
in order to, so as to, so that, objective, aim, purpose, achieve, maintain,
avoid, ensure, guarantee, want, wish, motivate, expected to…
```

List 3-9: Intentional search keywords for goal identification

```
improve, increase, decrease, reduce, enhance, enable, support, provide…
```

List 3-10: Amelioration search keywords for goal identification

Once a preliminary set of goals have been identified and validated by stakeholders, more goals can be identified by their *refinement* or *abstraction*. Finally, when a goal is refined enough to be performed by a single agent, the process of goal *operationalization* can take place (Lamsweerde, Darimont and Massonet, 1995; Van Lamsweerde, 2000; Van Lamsweerde, 2001).

3.4.2. Goal decomposition/refinement

Goal *refinement* or *decomposition* is a process of refining goal granularity by breaking down a higher-level goal into several lower-level goals in such a way that the latter contributes to the achievement or satisfaction of the former. It is the process of subdividing a set of identified goals into a logical sub-grouping so that system requirements can be more easily understood, defined and specified (Anton, 1996).

This informal, but effective, technique involves finding out sub-goals and requirements by asking "*HOW*" questions; that is, the sub-goals of *G* can be determined by asking "*How can G be satisfied?*", or "*Is this sub-goal sufficient or is there any other sub-goal needed for satisfying G*?" Furthermore, every time a *How* question has been answered, we can ask *How Else* again to explore other possible alternatives a goal can be refined (Van Lamsweerde, 2009, p. 311).

Goal refinement can be shown through *AND/OR* graphs called *goal diagrams*. An AND-refinement relates a goal to a set of sub-goals; it means that the parent goal can be satisfied by satisfying *all* sub-goals in the refinement An OR-refinement relates that *either* of the sub-goals can be seen as alternatives to achieving the parent goal (Van Lamsweerde, 2009).

An example is given in Figure 3-1. Here the soft goal of making the University of Twente an international university can be refined further as hard goals by asking How questions. We accomplish this, for example, by increasing marketing visits of International Office to other schools and by concretely increasing the enrollment by 10%, among others.

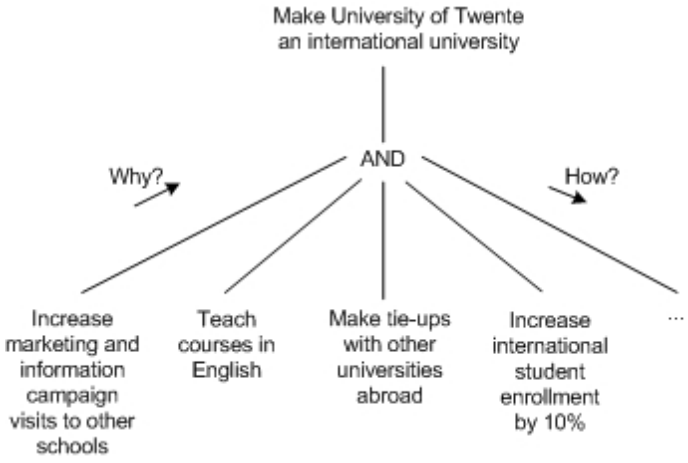


Figure 3-1: A sample business goal decomposition

The process is an iterative process and has to eventually stop refinement stops. This happens until all sub-goals are concrete enough to be assigned to individual agents of the system under development. This process helps in finding which sub-goals were overlooked but are needed to establish the parent goal (Van Lamsweerde, 2000; Van Lamsweerde, 2001).

3.4.3. Goal abstraction

Opposite to goal decomposition is goal abstraction, where new higher-level, more abstract, parent goals are explored based on identified decomposed sub-goals. Doing so helps finding out other sub-goals of the more abstract goal that were overlooked in the first place (Van Lamsweerde, 2000; Van Lamsweerde, 2001).

An obvious way of doing this informal, but effective, technique would be to keep asking "*WHY*" questions about goals already identified; that is, the parent goals of *G* can be found by asking "*Why should G be satisfied by the system*?", or "*Is there any other parent goal that G contributes to*?". As in the example in Figure 3-1, answering *Why* to the goal of teaching courses in English contributes to the parent goal of making the University of Twente an international university.

When eliciting system requirements, goal abstraction should be done within the system's subject domain, and should stop otherwise. Some authors equate goal refinement as both comprising decomposition and abstraction activities. Conversely, a goal cannot be abstracted further (i.e. we stop asking *WHY* questions) when the parent of a high-level goal cannot be satisfied solely by the cooperation of the system's agents only; that is, other systems not related to the system-to-be may needed to satisfy such higher level goal. Such high level goal is therefore out of scope of the system (Van Lamsweerde, 2009).

3.4.4. Goal operationalization

When goals are refined enough to the point where a single agent can perform it, goal operationalization may follow. Goal operationalization is the process of mapping goals to the *operations* that ensure their fulfillment (Van Lamsweerde, 2009, p. 427). It involves defining a goal until its sub-goals have enough detail to define its *operation* (Anton, 1996). Operationalization is a formal or informal technique to derive *pre, trigger and post conditions* on system *operations* so as to ensure goal fulfillment (Letier and Van Lamsweerde, 2002; Van Lamsweerde, 2009, p. 428).

A *precondition* must exist for the achievement of a goal to be possible; it *permits* the operation to be performed. When a *trigger* condition occurs, the operation *must* be performed to achieve the goal provided the precondition is true. The *postcondition* characterizes the *state*, or any other *condition*, of

the system after the goal is completed (Anton, 1996; Letier, et al., 2002; Van Lamsweerde, 2009, p. 428).

For example in a system for hiring employees, to achieve the goal *"available job position filled*", a precondition of "*applicant has applied*" must occur to trigger the "*fill job position*" action. The resulting state thereafter is "*employee matched position*".

3.5. Goals and business rules

This section relates goals to business rules. Before we do so, we first provide a brief introduction about business rules. According to the *Business Rules Group* (2000), a **business rule** is:

> **"A statement that defines or constrains some aspect of the business; it is intended to assert business structure or to control or influence the behavior of the business"**

Business rules are thus statements about guidelines and restrictions that define business operations. They are a set of permissible conditions that guide a business event so that it occurs in a way that conforms to desirable business outcomes. Thus, a business event fails if the rule governing it is not met (Von Halle, 2002). Business rules, like goals, may come from outside the organization such as laws or customs that guide the action of individuals and the organization. They may also come from within the organization such as business policies that together achieve some business goals (Taveter and Wagner, 2001). Business rules represent the knowledge that companies have about their business (Wagner, 2002).

However, more often than not, these rules may be scattered everywhere in the organization, and may not be even stated explicitly or made readily available. Sources of business rules may come from corporate charters, marketing strategies, pricing policies, customer relationship management practices, contracts and other legal documents. Worst, they are implicitly expressed or hidden as program codes in software applications that implement them (Wagner, 2002). This situation is not ideal especially in business domains where decisions and policies are intensive, extensive and dynamic. This is true, for example, in the finance, health, and insurance business sectors where a company may have hundreds, or even thousands of policies, procedures and definitions that interrelate together to guide business operations (Charfi and Mezini, 2004).

It is useful therefore that these rules are expressed, managed and updated efficiently and explicitly, independent of the rest of the applications that implement them. When business rules are logically (or even physically) independent from their platform specific implementation, they are better understood and made more easily accessible by business domain experts who specified them (Wagner, 2002).

A useful and salient property of business rules is that they can be specified in a near natural language such as English. Business rules are for business people, and when specified in a declarative way, as close as possible to natural language, business people can better understand, specify, validate their requirements (Nicolae and Wagner, 2008; Iacob and Jonkers, 2008). This is essential if business domain experts are to be involved in the design of the integration solution. Another property is that they can be made executable by software systems. When business rules are refined enough, they can be specified in a declarative language, transformed into executable rule expressions and deployed into some rule-based system. When, on the other hand, some business rules cannot or do not have to be operationalized or automated, they can remain expressed in natural language for the purpose of communication and documentation (Wagner, 2002).

3.5.1. Business rules basics

A business rule is usually written as a collection of *terms*, *facts*, and *rules* as shown in Figure 3-2 (Von Halle, 2002).

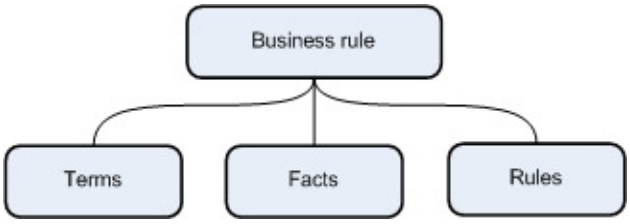


Figure 3-2: Business rule scheme (Von Halle, 2002)

Terms

A *term* is a noun or noun phrase whose definition is agreed upon by the business that use them. A term may define a concept (e.g. a student), a property of a concept (e.g. a student's ID number), a value (e.g. February), and a value set (e.g. Months). The Business Rule Group (2000) takes the definition of terms further according to its context: A *common* term is a term considered well known and unambiguous; whereas, a *business* term is a term whose meanings are interpreted in a certain business context. List 3-11 shows an example of a common term.

```
Bicycle
```

List 3-11: A common term

The meaning of the business term in List 3-12 may vary to some degree depending on how the term is applied in some context, for example, in a university library book reservation system, a hotel guest reservation system, airline company passenger reservation system, etc.

```
Reservation
```

List 3-12: A business term

Facts

A *fact* is a statement that connects terms, through prepositions and verb phrases, into sensible, business-relevant observations. They are used to establish and define the relations and roles of each term (Business Rule Group, 2000). Terms and facts provide semantics to the rules. List 3-14 shows an example of a fact.

```
A student has an ID card
```

List 3-13: A sample fact

Rules

While terms and facts are statements that assert some basic information value, I allow the discovery of new information or guide decision making by applying logic or computation to the information values. Rules eventually provide executable logic that uses some information as input and creates either a new action or information as output. List 3-15 shows an example of a rule.

```
A student can only reserve a book after presenting an ID card
```

List 3-14: A sample rule

3.5.2. Business rules classification

This section provides a much larger classification of business rules based primarily on their intended use. There is no universally accepted business classification scheme, and companies are in fact encouraged to classify business rules according to their needs (Von Halle, 2002). We briefly identify three of the major categories: *Deductive*, *Integrity*, and *Reaction Rules*.

3.5.2.1. Deductive/Inference/Derivation rules

Deductive rules establish new facts that results when a rule evaluates to true. Based on existing knowledge and logical dependencies between facts, new knowledge can be derived (or inferred). The derivation of new knowledge is a result of some logical inference or mathematical calculation (Braye, et al., 2006). These types of rules capture heuristic (experience-based) knowledge without explicitly storing such information – allowing them to be extracted on demand (Rosenberg, Nagl and Dustdar, 2006). Deductive rules have been applied for a long time in health care that uses rule-based expert systems, especially in the diagnosis of diseases.

A simple example of a derivation rule is shown below:

```
If an animal barks, then it is a dog
```

List 3-15: A sample derivation rule

3.5.2.2. Integrity Rules

Integrity rules are assertions that need to be satisfied in all stages of a system. Two types of such rules can be distinguished: those with *state* or *process* constraints. State constraints must hold valid at any time in the system, while process constraints describe the valid state transitions in the system (Rosenberg and Dustdar, 2005). Taveter and Wagner (2001) also define integrity rules as *integrity constraints*.

A sample integrity rule with state constraint is:

```
Only bona fide students of the university can borrow books
```

List 3-16: A sample integrity rule with state constraint

A sample integrity rule with process constraint is:

```
A student must first select a book to borrow before he can check it out
```

List 3-17: A sample integrity rule with process constraint

3.5.2.3. Reaction Rules

Reaction Rules cause actions to be performed when a certain event occurs or a certain condition is satisfied. Such action could be manipulated by one or more business objects, the occurrence of another business activity, the action of a business actor, etc. (Rosenberg, Nagl and Dustdar, 2006). They state conditions that describe when actions must occur in response to some event. They usually consist of event conditions, a state condition (or precondition), an action term, and a state effect (or post-condition). They have the general format of `if-Event|Condition-then-Action|Effect` (Wagner, 2002). As such these rules can also be called *stimulus-response rules*, *action rules*, or *Event-Condition-Action* (ECA) *rules* (Rosenberg and Dustdar, 2005).

```
If insurance claim is greater than 10000 €,
  then send it to the Manager for approval
```

List 3-18: A sample reaction rule

Production Rules

A special case of reaction rules are *production* rules. Production rules have an `if-Condition-then-Action` format – compared to derivation rules which have an `if-Condition-then-Conclusion` format.

Production rules can be simulated using *production rule systems* as they do not explicitly refer to events. In such systems, facts are *asserted* into working memory (usually a RAM memory which holds a collection of instantiated facts). A given rule executes (or fires) based on a set of conditions and the given state of the facts in the working memory. Production rules can simulate derivation rules, following the `format if-Condition-then-assert-Conclusion`, using a special function called `assert which` changes the state of a production rule system by adding facts from working memory (Wagner 2002). An example of a production rule system is Java Expert System Shell (Jess).

```
(assert (animal-is Sheena))
(assert (barks Sheena))

(defrule is-dog
    "if an animal barks, then it is a dog"
    (animal-is ?x)
    (barks ?x)
     =>
       (assert (is-a ?x dog))
 )
```

List 3-19: A sample production rule in Jess

List 3-19 shows an example of a production rule in Jess simulating a derivation rule. The rule "`is-dog`" fires when the fact "`animal-is Sheena`" and "`barks Sheena`" are asserted into working memory (which in this case is true). The "`then`" part of the rule executes which simply asserts a new fact in working memory and thus concludes "`is-a Sheena dog`". A more thorough introduction to Jess is given in Section 5.2.4.

3.5.3. Implementing goals as business rules

The relationship business rules to goals and can also be viewed as a form of goal operationalization. As we have described in Section 3.4.4, goal operationalization involves refining the goal to the point where a single agent can perform or satisfy it. Goal operationalization thus requires the mapping of goals to the *operations* that ensure their fulfillment. These operations have constraints in the form of pre, trigger and post conditions to ensure goal fulfillment. Iacob, et al. (2009) argues that goals are thus made operational through their constraints. Business rules are essentially constraints in that they provide permissible conditions that guide desirable business outcomes (von Halle, 2002). Thus, we can see that goals can be operationalized as business rules through the constraints (or conditions) that satisfy their fulfillment.

Kardasis and Loucopoulos (2005) argue that business rules can be derived from an organization's business goals. An organization may have strategic goals that describe high-level business vision and objectives. These strategic goals can be refined into one or more operationalized goals that describe a set of actions to accomplish the strategic goal (also called a tactic). An organization can have some policies that express some specific or quantified constraint to accomplish tactic. These policies can be expressed as business rules. Finally, these rules are later translated in the form of Event-Condition-Action rules and deployed in some rule based system. Thus, an organization's goals may be fulfilled or satisfied if the business rules that implement them are satisfied. This relationship can be depicted graphically in Figure 3-3.

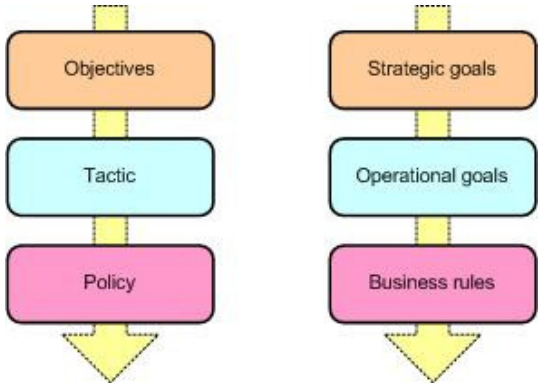


Figure 3-3: From goal to rules (Kardasis and Loucopoulos, 2005)

Taking the example of Kardasis and Loucopoulos (2005):

- To support evaluation of responses to Request for Proposal is an enterprise goal at the strategic level (strategic goal).
- To identify an RFP response that represents an acceptable technical solution at the lowest possible price (operational goal).
- To accept an RFP response, if the financial offer is the lowest among all and the technical offer score is above 80 percent (business rule)

They thus treat a business rule as either the effect of goal operationalization or the rationale behind the operationalization of a business rule. More specifically, the relationship between goals and business rules can be described in the following manner:

- "Goals reveal rules that are enforced by them;
- Goals reveal rules that create opportunities for their fulfillment; or
- Goals may reveal rules that hinder their fulfillment".

3.5.4. When to use a rule-based solution?

Rudolph (2008) proposes the following points to consider when choosing rule-based technologies as a solution to the design of software systems.

- Choose a rule-based solution when the system under development involves significant conditional branching or decision-making. This is evident by the too much use of if-then-else statements in the code that mostly represent business rule validation; otherwise, if the system requires intensive computations or lookups over a database, and not much conditional branching required, then don't use rule-based solutions.
- Choose a rule-based solution especially when the rules that specify business conditions are complex. If you can't state rules in your system, then don't use a rule-based solution. If the system has two or a few rule conditions or some shallow nested if-then-else statements, then using a rule-based solution would be an overkill. Otherwise, if there are a number of rules that need to be written or with very deep if-then-else statements, choosing a rule-based solution would be wise.
- Choose a rule-based solution when there is certainty that rules will change reasonably over a period of time; otherwise, if rules remain fixed or static, the flexibility and overhead of using a rule-based approach is usually not needed.

- Corollary to the previous point, choose a rule-based solution when the code that encapsulates the rules or the finished product need to be maintained over a period of time. Otherwise, if the rules encoded in the finished product were a one-shot effort and does not require maintenance, then a rule-based solution is not necessary.
- Rule-based solutions that use Rule engines like Jess favors speed over memory usage. When using Rule engines, expect that resulting systems are usually memory intensive. Choose a rule-based solution when the requirements favor speed over memory usage.
- Business-wise, the cost involved in using rule-based solutions and the purchase of a rule engine usually includes licensing fees for development and deployment of the engine and training cost of developers and users. Furthermore, financial analysts have shown that it takes about one year to break even monetarily when investing on rule engine technologies. So that do not choose a rule-based engine when the project schedule or project lifecycle cannot accommodate the cost of using a rule engine, the company cannot wait for one year to begin to see significant return of investment (ROI), and no funds can be afforded to train developers and users.

3.6. Goal modeling approaches

Goal modeling is central in goal-driven approaches in RE (Rolland and Salinesi, 2005). It plays an important role in supporting heuristic, qualitative or formal reasoning in RE sub-processes such as requirements elaboration, consistency and completeness checking, alternative selection, evolution management, etc. (Van Lamsweerde, 2001).

A *goal model* shows how goals are refined, linked, and shown to contribute to one another. It captures alternative ways of refining goals and identifying conflicts among them. It also describes responsibility links between goals and agents, links between obstructions and operationalization. The model can also specify certain properties of a goal such as name, precise specification, type, category, priority, source, etc (Van Lamsweerde, 2009, p. 294).

For business domain experts, and others who use business models, managing the complexity of requirements is important. Goal modeling allows them to explicitly represent what the business needs to do in a form understandable to them. Moreover, the process of creating goal models is in itself a learning process; that is, business domain experts are able to gather, assume, and test their goals while in the process of creating them. This stresses the notion that goal modeling is a crucial prerequisite in the design and management of good business processes (Kueng and Kawalek, 1997).

This section provides a brief overview, key concepts and observations of the goal modeling approaches reviewed in this research: KAOS, i*, BMM, and ARMOR. We use the extensive comparisons between goal modeling languages by some authors (Iacob, Rothengatter and Van Hillegesberg, 2009; Quartel, 2009a; Quartel, et al., 2009c; Lapouchnian, 2005; Engelsman, 2008).

We look into how these goal-driven approaches can be used to specify the requirements for the design of the mediation solution; specifically, in terms of how they provide an opportunity for requirements, modeled as goals, to be used as basis for specifying the *Enterprise Architecture* design of the Mediator. An Enterprise Architecture describes an organization's structure, processes, applications, systems, and technology in such an integrated way using methods, techniques, models, and visualizations through a set of principles, methods and models. It provides a high-level holistic view of the enterprise that brings together information from formerly disassociated domains to foster understanding by various stakeholders of the domain (Lankhorst, et al., 2005).

3.6.1. i*

Overview

The *i** framework[4] (pronounced eye star) is the result of Eric Yu's PhD dissertation at the University of Toronto, Canada (Yu, 1995; Yu, 1997). The primary focus of i* is to model and analyze requirements at the early phases of requirements engineering by putting emphasis on the rational of the underlying systems (i.e. the *why*) rather than specifying *what* the system is supposed to do.

Key concepts

i* takes an agent-oriented requirements engineering approach; that is, it uses the notion of an *intentional actor* as its core concept. These actors possess intentional properties such as goals, beliefs, abilities, and commitments that are used to reason about their strategic relationships. Actor dependencies provide the exploration of opportunities and threats.

i* models requirements using two types of models: *Strategic Dependency* (SD) and *Strategic Rationale* (SR). The Strategic Dependency model describes dependencies or intentional relationships among actors in an organizational context. Such dependencies provide a better understanding of the *whys*. Figure 3-4 shows an example of the SD model taken from Deng (2006) where the Health Care Provider actor *depends* on the Claims Processing Unit for making accurate payments (a soft goal).

The Strategic Rationale model describes how stakeholder interests or rationales are made explicit, and how system or environment configurations can impact these interests. Essentially, an SR model details the SD model by looking into how actors model internal intentional relationships which include such *intentional elements* as goals, tasks, resources and soft goals. These elements are linked by *means-end relations*, *task decompositions,* or *contribution relations* (representing how a sub-goal or task contributes to a soft goal). Figure 3-5 shows an example of a Strategic Rationale model between the dependency of Health Care Client and Claims Processing Unit actors to ensure privacy of personal and health care information as shown in the SD model of Figure 3-4 (figure taken from Deng, 2006). Here, the soft goal Privacy helps contribute to the quality of service offered by the Claims Process Unit actor.

The i* framework has been used as basis for TROPOS[5] to extend requirements modeling to the later phases of the software lifecycle.

Observations

The intentional concept of i* has been applied in the context of EA as shown by Yu, Strohmaier and Deng (2006) and Deng (2006). Their approach, however, has largely focused the *process of Enterprise Architecture construction* which starts by articulating an architectural vision, develop an as-is Enterprise Architecture, listing business problems and root causes, develop alternative Enterprise Architecture configurations, selecting a configuration, and finally completing the target Enterprise Architecture. From the last step, however, the process does not provide guidance as to how the target Enterprise Architecture is to be modeled and implemented by the underlying application and physical systems; including the relations between these systems. Modeling thus stops at this stage.

[4] http://www.cs.toronto.edu/km/istar/
[5] http://troposproject.org/

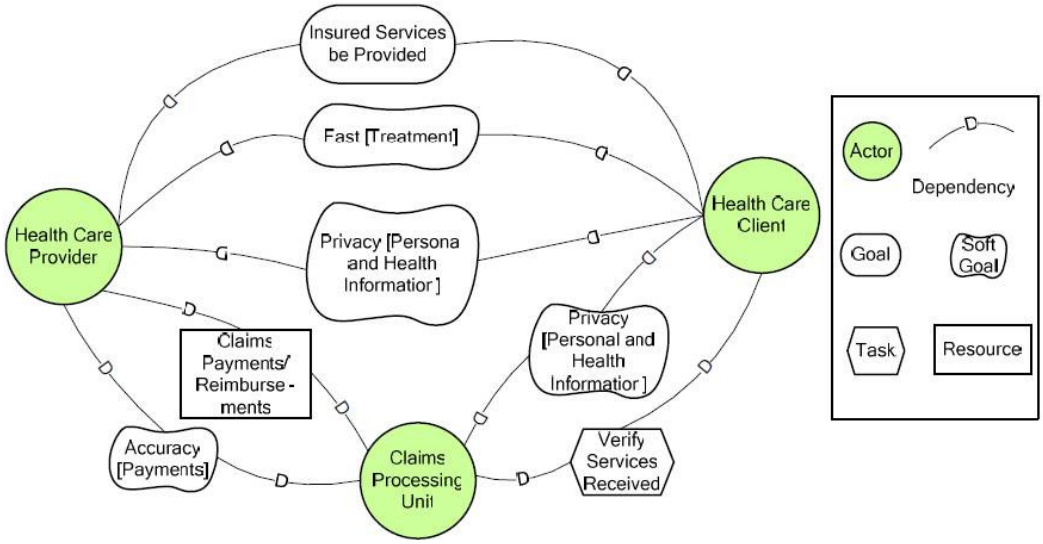


Figure 3-4: A sample Strategic Dependency model (Deng, 2006)

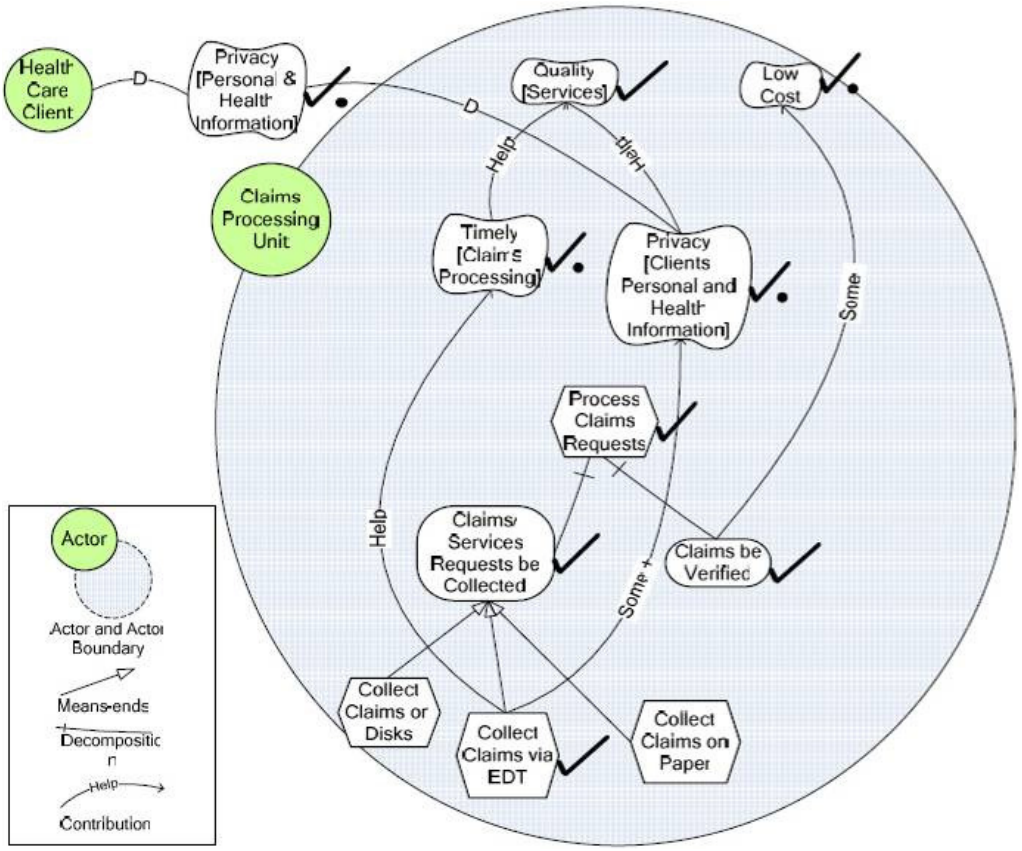


Figure 3-5: A sample Strategic Rationale model (Deng, 2006)

3.6.2. KAOS

Overview

KAOS (*Knowledge Acquisition in autOmated Specification*) (Van Lamsweerde, 2009) is a methodology for requirements engineering initiated by Axel van Lamsweerde of the Department of Computing Science and Engineering, Université catholique de Louvain, Belgium, which started as a cooperation with the University of Oregon in 1990. KAOS takes a strong formal approach to goal modeling; that is, temporal logic and refinement techniques are used to rigorously define goals to meet the requirements of the system-to-be. Because of its formal background, its powerful feature comes from its ability to analyze and reason about goals, their refinement, validation, operationalization, completeness and traceability checking, and conflict and risk analyses.

While i* focuses on modeling requirements at the early phases, KAOS focuses more on the later requirements engineering phase. Although KAOS, like i*, allows modeling of motivations of requirements, it does not give too much emphasis on the intentions of actors like i* does (Quartel, 2009a).

Key concepts

A *goal* which is "a prescriptive statement of intent that the systems should satisfy through the cooperation of its agents" is the key concept in KAOS. An *agent* can be a device, human user, the system under consideration or the system to be developed that performs some role to satisfy the goal.

Higher-level goals can be defined into lower-level goals indicating how the latter satisfy the former through refinement techniques using AND/OR constructs (see Sections 3.4.2 and 3.4.3). When goals are refined enough to be assigned to a single agent, they can be modeled as *requirements* or *expectations*: a goal that can be assigned to a system-to-be (i.e. an automatable goal) is called *requirement*, while a goal that can be satisfied by the environment of the system-to-be is called an *expectation* (i.e. a goal that cannot be enforced by the system-to-be). KAOS also provides constructs to model conflict relations between goals (i.e. how a goal restricts the achievement of another goal) using the *obstacle* construct. Finally, KAOS can also model elements of the problem domain: a *domain hypothesis* describes properties of the problem domain that expected to hold, while domain invariants describe properties that always hold.

Figure 3-6 provides a summary of the concepts and notations of KAOS, and their relation. The *goal modeling* part captures all the stakeholder's needs and wishes including the identification of obstacles, threats, and faults. The *responsibility modeling part* describes distribution of system responsibilities including the capabilities of agents. The *object modeling* part captures the conceptual objects of the domain-specific concepts. Finally, the *operation modeling* part describes system operations, their features and their links to the goal, object, agent, and behavior models (Van Lamsweerde, 2008).

Figure 3-7 shows an example of a goal model of an elevator system in KAOS. The main goal *service requested* is refined into three sub-goals that which must be satisfied to satisfy the main goal. The *system designer* is made responsible to the achievement of the requirement *user interface provided*. The expectation service requested through the interface is assigned to the *user* agent.

Observations

KAOS has been applied to the design of software architectures where a process is described to generate from high-level software requirements, represented as goals, to software specifications, and then to a dataflow architecture (Lamsweerde, 2003). KAOS has also been to shown to derive a behavior architecture, through event-based transition systems, which is derived from software requirements represented through goals (Letier, Kramer, Magee and Uchitel 2006).

As KAOS has a strong formal foundation, the resulting architectures were described formally as well. This is good for critical systems that require a high-degree of precision. In the context and at the level of EA, however, we observe this is very low level to be appreciated architecturally. We are more interested at how software elements and their structural relations are described in an abstract manner through their public interfaces and not their internal details (Bass, Clements and Kazman, 2003). Also, it is not clear how the described dataflow and behavior architectures are related to each other.

Finally, KAOS has a limited and an unofficial set of notations to describe architectural elements. It does not also allow how models at one layer of abstraction can be modeled against another layer (e.g. matching software elements from both the application and technology layers).

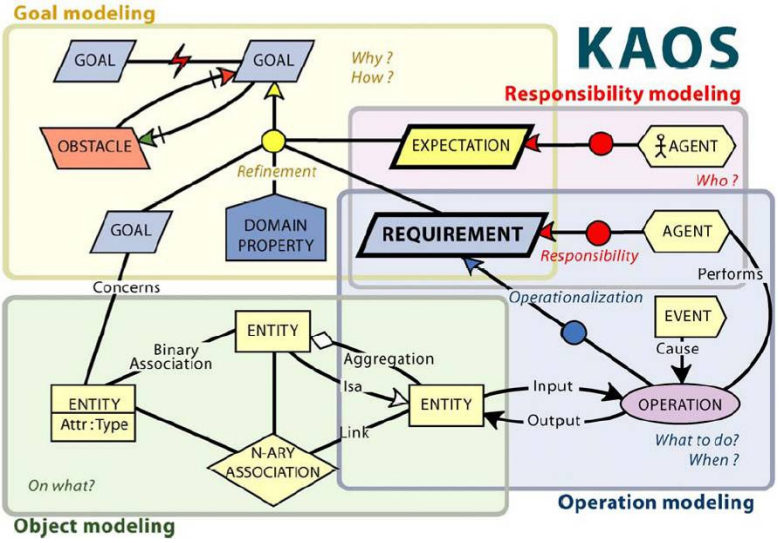


Figure 3-6: Summary of KAOS concepts and notations (Respect-IT, 2007)

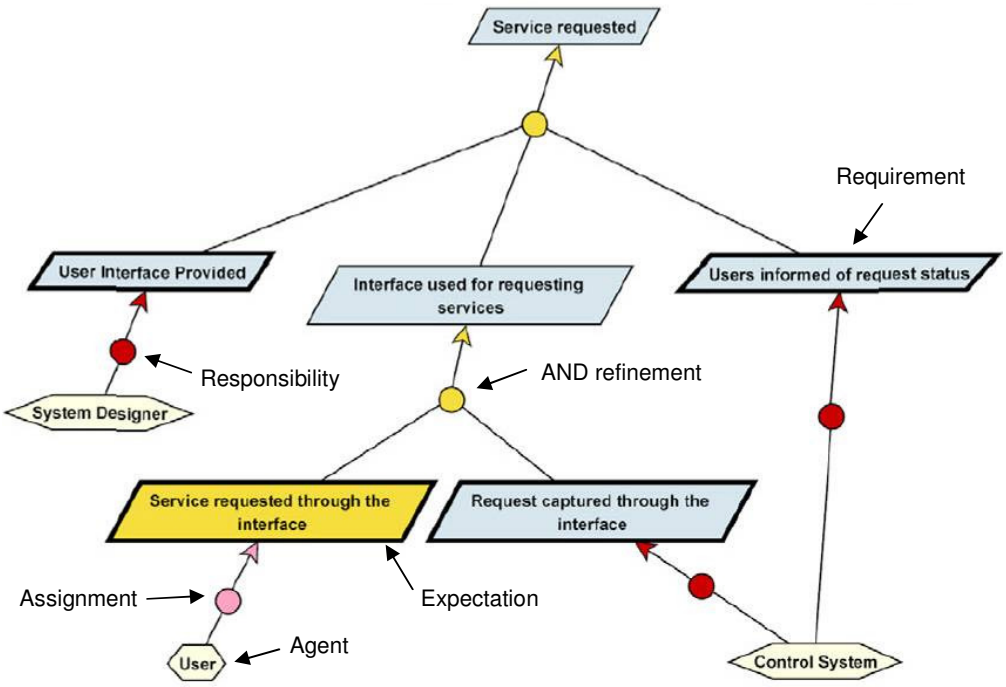


Figure 3-7: A sample goal model in KAOS (Respect-IT, 2007)

3.6.3. BMM

Overview

The *Business Motivation Model* (BMM) was initiated by the Business Rules Group in 2005 and is now currently adopted by the Object Management Group since 2008 with a Request for Comment (RFC) status (Object Management Group, 2008b). BMM provides a set of meta-models essential to business governance. It provides structure to develop, communicate and manage business plans in an organized way. It identifies factors to motivate, identify and define elements of the business plan, including the relation of these elements. Essentially, BMM defines business elements *before* system design or technical development starts. Translating the business plans to technical specification and implementation is thus out of scope.

Key concepts

At the heart of BMM is the *motivation* concept which allows an enterprise to describe *why* they have chosen to take a business activity; that is, they should be able to describe the results that these activities want to achieve. The meta-model constructs of BMM shown in Figure 3-8. We now discuss them briefly:

- *Ends* describe what the enterprise wants to be (e.g. changing how it does its business to explore new market opportunities). Ends do not say how the enterprise will achieve them. In BMM, Ends can be categorized into a general *Vision* (the image that the organization wants to be or become) and the specific *Desired Results*, which is composed of *Goals* (an End defined qualitatively and set in long term) and *Objectives* (a step towards the Goal).
- *Means* are actions that an enterprise decides to do to become what it wants to be (i.e. to achieve the Ends). They constitute some devices, capabilities, regimes, techniques, restrictions, agents, instruments to achieve the Ends. In BMM, Means are organized into *Mission* (ongoing operational enterprise activity), *Courses of Action* (activities that the enterprise has decided to do), and *Directives* (doable and undoable activities, including limits about how activities should be done).
- *Influencers* are those that can cause changes (or the absence thereof) while the enterprise implements its Means to meet its Ends. They may come from within the enterprise (*Internal Influencers*) or outside the enterprise boundary (*External Influencers*). Modeling influencers allows enterprises to describe how they can react appropriately to change.
- *Assessments* describe how an enterprise should judge or react to the influences (created by the Influencer) that affect the enterprise in its ability to employ its Means or achieve its Ends. BMM suggests the use of SWOT (Strength, Weakness, Opportunity, Threat) as a sample approach for making assessments.

Observations

BMM provides a broad and holistic framework for modeling business motivations of an enterprise. As it takes a strong emphasis on business perspective analysis, its intended audience is primarily business people (albeit it does not provide a graphical notation for modeling requirements). BMM is an open standard and is easily integrated with other OMG standards; for example, SBVR (Object Management Group, 2008a) which may prompt its wide adoption. For our purposes, however, we find BMM having too broad of a scope; that is, much emphasis is given to modeling the business desires of an enterprise without prescribing which requirements will be implemented by underlying software systems and how they should be implemented (or should they be). As BMM focuses on early the phases of requirements engineering, it does not prescribe a methodology or architecture to translate business requirements into their technical implementations. In fact, some authors do not consider BMM as a true requirements modeling language (Quartel, 2009a).

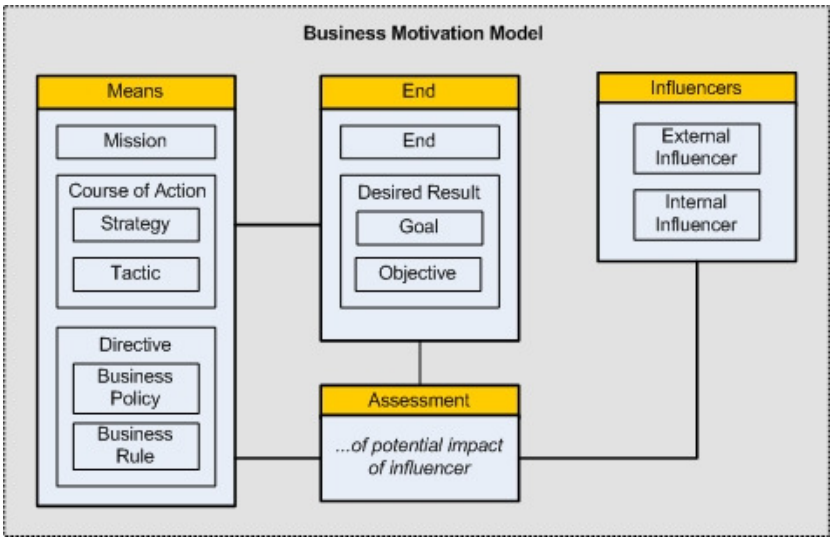


Figure 3-8: BMM meta-model relations

3.6.4. ARMOR

Overview

Architectural Modeling of Requirements (ARMOR) is a *goal modeling language* in *engineering requirements* for the design of EA developed by Novay in cooperation with BiZZDesign, the Dutch Ministry of Economic Affairs, the Dutch Tax Administration and the University of Twente as part of Novay's BServed project[6] (Quartel, 2009a; Quartel, et al., 2009c). Current EA modeling techniques have largely focused on answering "*what"* the enterprise should do. ARMOR adds to this by focusing on the "*why*"; that is, it seeks to explicitly represent the reasons for designing the architecture in terms of its motivations, rationale and requirements. ARMOR is not a yet-another-goal-modeling-language invention. Its language constructs are largely derived from and aligned with current goal modeling languages such as KAOS, i*, and BMM applied in the area of RE in EA by extending ArchiMate[7] (Lankhorst, et al., 2005) modeling framework for EA.

The design of ARMOR is largely motivated by addressing the importance of RE in understanding, structuring, modeling and analyzing how business requirements can be related to IT requirements for improved business-IT alignment and requirements traceability in EA. It also places strong emphasis on ease of use by providing a lean, easy-to-learn, limited set of goal modeling concepts without compromising its expressiveness. It provides a set of abstract and concrete syntax as with most goal modeling languages described earlier.

Recently, ArchiMate has been adopted by The Open Group[8], a vendor- and technology-neutral consortium, which seeks to integrate information within and between enterprises based on open standards and global interoperability. The original developers of ArchiMate represented a broad partnership from business and government (ABN AMRO, ABP, Dutch tax department, Ordina) and science (Radboud Universiteit, CWI, Leiden University), led by the Novay[9] (formerly, Telematica Instituut).

[6] http://www.novay.nl/okb/projects/bserved/4520
[7] http://www.archimate.org/
[8] http://www.opengroup.org
[9] http://www.novay.nl/

Key Elements

ARMOR extends the ArchiMate framework by adding motivation and *meaning aspects*, and the *value layer*. A *layer* in ArchiMate represents successive abstraction levels at which an enterprise can be modeled (i.e. business, application, and technology layers); whereas, an *aspect* represents different enterprise concerns (i.e. actors, behavior, and information, and their relations). The intersection between the layer and the aspect is called a *viewpoint* which represents a certain perspective on the enterprise which is of interest to various stakeholders. A viewpoint, in turn, can be used to describe concepts that may cover several layers and aspects. This is modeled orthogonally as *domains* which covers a set of concepts of a particular viewpoint.

ARMOR adds value and meaning aspects to ArchiMate. The *value* layer represents how the enterprise can offer value to the customers through its products and services. The *meaning* aspect represents the semantics of the enterprise (architecture) artifacts such ontologies used by the enterprise and its customers. The *motivation* aspect represents goals and intentions of the enterprise. ARMOR also adds several domains as a result of the additional viewpoints created by the intersection of the value layer and meaning aspect: The *stakeholder* domain models stakeholders and their concerns including the assessments of these concerns. The *principles* domain covers the vision, mission, strategies, policies, principles and guidelines of the enterprise, and the *requirements* domain which cover the goals, requirements, and expectations that constrain the design of the Enterprise Architecture.

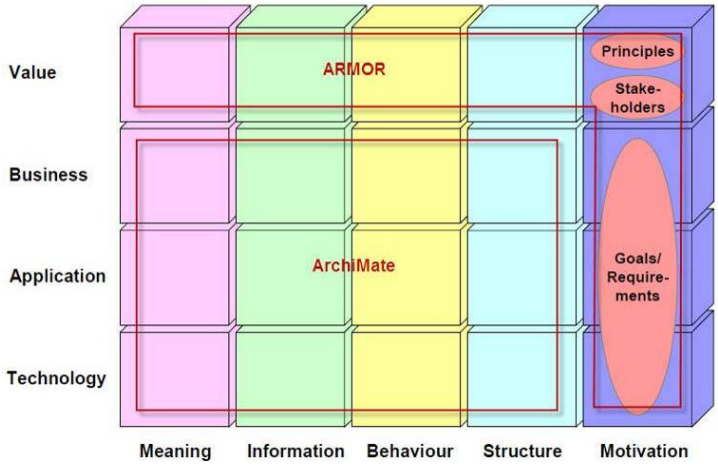


Figure 3-9: Extending ArchiMate with ARMOR (Quartel, et al., 2009c)

Observation

As ARMOR builds on existing enterprise (architecture) modeling frameworks such as ArchiMate, it is able to take advantages of the benefits of the latter (e.g. business-IT alignment, requirements traceability, etc). In particular, ARMOR provides a way to describe how goal models are supported by architectural services that realize them at the various layers of the enterprise. It uses the concept of *service provisioning* where a layer exposes functionalities of components for use and realization by other components in different layers; the objective of which is to achieve better business and IT alignment.

ArchiMate provides a set of coherent architectural modeling constructs that specify, describe and relate architectural components unambiguously. The *integrated* modeling of architectural domains allows better understanding from both business domain experts and IT experts (Lankhorst, et al., 2005). With ARMOR and ArchiMate, we are able to achieve *continuity of modeling* from requirements layer down to their business, technical, and finally physical implementations which is absent from other goal modeling approaches described earlier. Thus, ARMOR's concept of relating goals and the Enterprise Architecture could be useful for describing our service mediation solution. We shall therefore investigate this in our thesis.

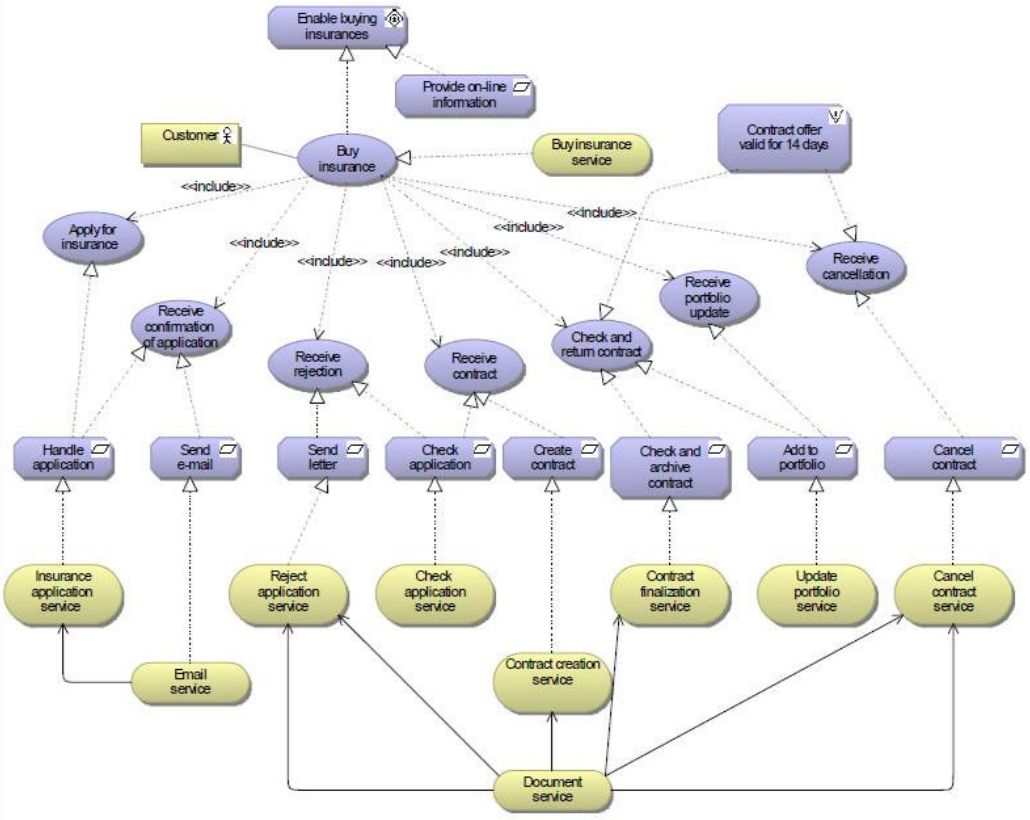


Figure 3-10: A sample goal model in ARMOR (Quartel, et al., 2009c)

3.7. Chapter summary

Goals are prescriptive statements that describe the motivations of the mediation solution. They provide a way to specify in a disciplined and structured manner the objectives (or the "whys") of the system at various levels in the enterprise. The cooperation of some agents (e.g. organizational actors, operators, end users, etc.) is needed to satisfy the achievement of the goal(s).

Goal classified according to strategic vs. operational goals, functional vs. non-functional, soft vs. hard goals. Strategic goals are more abstract objectives that organizations want to achieve. They cannot usually be performed by a specific process but nevertheless important for the organization. Operational goals define the desired state to be achieved which can be accomplished by a specific process. Functional goals underlie services that a system is expected to deliver; that is they describe what the software does. Non-functional goals refer to expected system qualities such as security, safety, performance, usability, flexibility, customizability, interoperability, and so forth. Soft goals are goals whose satisfaction cannot be established in a clear-cut sense. Hard goals are goals whose satisfaction can be established through verification techniques.

GORE provide the ability to elicit, elaborate, structure, specify, analyze, negotiate, document, and modify the integration requirements in a structured manner. GORE activities such as goal identification/analysis, goal decomposition/refinement and goal identification techniques are used Goal analysis is the process of exploring gathered documentation for the purpose of identifying, organizing and classifying goals. Goal decomposition/refinement is a process of refining goal granularity by breaking down a higher-level goal into several lower-level goals in such a way that the latter contributes to the achievement or satisfaction of the former. Goal

abstraction is the process of exploring new higher-level, more abstract, parent goals based on identified decomposed sub-goals. Goal operationalization is the process of mapping goals to the operations that ensure their fulfillment

GORE provides a natural way for structuring complex specifications, reason the existence of a specific requirement, provide a starting point for the identification of requirements, connect requirements to design, analyze potential risks, manage conflicts among requirements, offer various alternative, lends itself to requirements traceability and evolution support.

Business rules can be derived from goals. Business rules can be treated as the refinement of a goal in a way that it constraints some aspects of its satisfaction.

Business rules are statements about guidelines and restrictions that define business operations. Properties of business rules include their ability to be stated in near-natural language and be made executable.

Business rules may be classified according to deductive rules, integrity rules, and reaction rules. Deductive rules establish new facts that results when a rule evaluates to true. Integrity rules are assertions that need to be satisfied in all stages of a system. Reaction Rules cause actions to be performed when a certain event occurs or a certain condition is satisfied. A special case of reaction rules are production rules where they follow the `if-Condition-then-Action` format – comparing to derivation rules which have an `if-Condition-then-Conclusion` format.

Among the surveyed GORE approaches (i.e. i*, KAOS, BMM, and ARMOR) we find ARMOR to be the most suitable to specify the integration requirements because of its tight connection with the Archimate enterprise modeling framework. ARMOR + Archimate allow us to take advantages of the benefits of better business-IT alignment and requirements traceability. In particular, ARMOR provides a way to describe how goal models are supported by architectural services that realize them at the various layers of the enterprise. Furthermore, the integrated modeling of architectural domains allows better understanding from both business domain experts and IT experts. Finally, we are able to achieve continuity of modeling from requirements layer down to their business, technical, and finally physical implementations which is absent from other goal modeling approaches.

4

Service mediation

This chapter discusses concepts related to service mediation as used in this thesis. We study current service-oriented mediation approaches through a state-of-the-survey, and describe how these approaches allow business domain experts to participate in the design of the integration solution.

The structure is as follows: Section 4.1 describes basic concepts related to service mediation. Section 4.2 presents the result of a survey of service mediation approaches. Section 4.3 summarizes this chapter.

4.1. Definitions

A myriad of concepts on mediation exists in computer science literature today – ranging from object-oriented design patterns, network communication topologies, database mediator hubs, to agent-oriented matchmaking and brokerage techniques. This section describes what we mean by service mediation as it is used in this thesis.

In Service Oriented Architectures, Papazoglou (2007) defines a **service** as:

> **"A self-contained, platform-agnostic computational element that supports rapid, low-cost and easy composition of loosely coupled distributed software applications".**

They offer a functionality that can be described, published, and discovered, which can then be further assembled to form more complex systems. They integrate systems that are not originally intended to integrate easily with other existing systems without having to be tightly coupled.

A service can also be viewed as a set of related *interactions* between a system and its environment which produces an *effect*. This effect has a *value* to the entities of the environment where the interactions are taking place. These interactions are realized by the exchange of *messages* between the system and its environment. The effect is created when a system sends a message to its environment; for example when system provides information to or changes a state in its environment. These messages consist of *data* and *behavior* properties. The data properties represent values that describe what the message is all about. The behavior properties describe the ordering of the messages in its environment. Thus, so that systems can interact effectively, they must have a common understanding of the data in the message and a common expectation of the effect of the message (Pokraev, Reichert, Steen and Wieringa, 2005).

Several challenges arise when systems attempt to *interoperate*; i.e., when different systems exchange information and use that information effectively (IEEE, 1990). For one, enterprises have invested on large old systems whose functionalities are not originally meant to be exposed and used by other systems – a product of the monolithic architectures in the early days. These heterogeneous systems, built on different hardware or software platforms that follow different standards, have difficulty in performing the operations offered by another system because of different message execution sequences, message formats, or message meanings.

In this thesis, we provide a solution to this interoperability problem through *service mediation*. Service mediation is ideal when two or more existing systems want to collaborate through their services, which to some extent are complementary, to produce a desirable effect. This collaboration, however, is hindered because of differences in the way information is represented or interpreted, or the order in which messages are exchanged. As a consequence, these systems cannot directly communicate with each other. One solution would be to make either of the systems redefine its services, for example, by reprogramming encapsulated business logic for the purpose of collaboration; this, however, can be very expensive. This is also not ideal when these systems need to be dynamic since similar changes may have to be done to accommodate collaborations with new systems, and is therefore not practical.

In order to make these systems interoperable, a 'third' or intermediary system is needed to handle mismatches in the data and behavior or *processes* properties of the messages without causing either system to change their services. We call this third system a *Mediator*. Figure 4-1 shows a Mediator *M* matching System A's *S1* requested service by composing System B's *S3* and *S4* provided services to produce a desirable effect in the collaboration. The example below shows a one-to-one mediation between systems. One-to-many mediation is also possible.

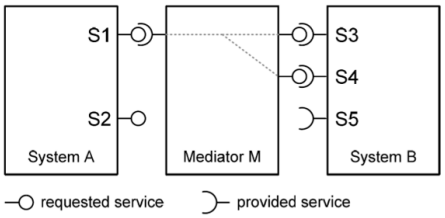


Figure 4-1: Service mediation (Quartel, et al., 2008a)

In this thesis, we adopt the definition of **service mediation** of Quartel, et al. (2008a) as:

> **"To act as an intermediary agent in reconciling differences between services of two or more systems."**

Service mediation involves reconciling two types of differences or mismatches: *process* and *data*. Process mismatches occur when systems use services that define different messages or different ordering of message exchanges. For example, consider two Systems *A* and *B*. System A intends to deliver two messages *M1* and *M2* in the exact sequence whereas System B expects it the other way around. The Mediator *M* solves this mismatch by reordering the messages of System A to the expected result of System B. This mediation pattern is called *Message Reordering* shown in Figure 4-2. For more types of process mismatches, we refer you to Pokraev and Reichert (2006).

Data mismatches occur when systems use different information models (or vocabularies) to describe the messages that are exchanged by their services. The aim is to preserve semantics between such incompatibilities. Figure 4-3 shows an example of a data mismatch scenario provided by the Semantic Web Service (SWS) Challenge[10]. As part of requesting payment initiation, System A sends three data items (i.e. `iso20022:BIC`, `iso20022:IBAN`, and `iso20022:Ccy`) formatted using an ISO-based standard to System `B` formatted using its own structure. In order for System A's message to be useful to System B, the equivalent meaning of the messages between systems must first be established. The Mediator handles this required data

[10] http://sws-challenge.org/wiki/index.php/Main_Page

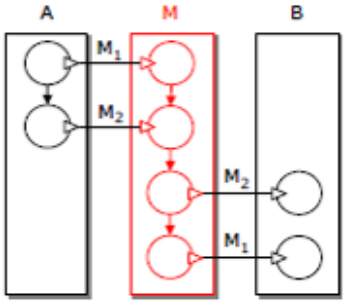


Figure 4-2: Message reordering

mediation. For example, `iso20022:BIC` is matched to `p:swiftCode` which `both` refer to the same bank identification number. This example shows a naming conflict mismatch (also called *Synonyms*) where attributes are semantically alike (i.e. same `AB10009` message) but have different names (i.e. Systems A calls it `BIC`, while System B calls it `SWIFT`). For more types of data mismatches in service messages, we refer you to Nagarajan, Verma, Sheth and Miller (2007). Resolving semantic heterogeneities between data elements is a difficult process. A large part of today's approaches is largely manual in nature where business domain experts come together to first decide on the equivalent semantics of the elements before they are implemented automatically using software components.

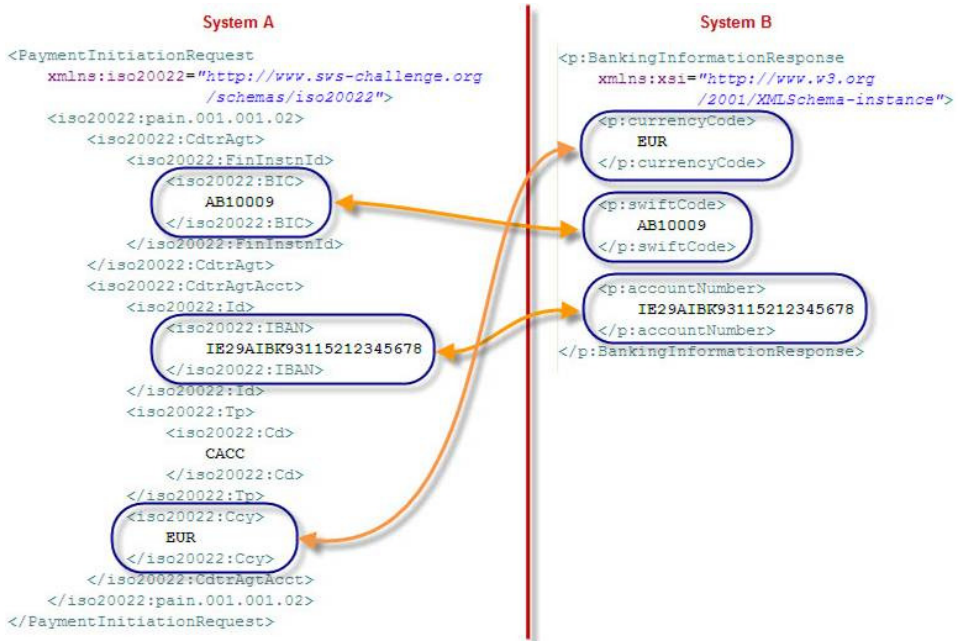


Figure 4-3: Synonym type message heterogeneity

Assumptions

The main focus of this research is solving process mismatches. We assume that a common message format to structure the messages is already in place; i.e., although services do not match directly, they agree and use the same message format to exchange messages. Thus data mediation is no longer necessary. Therefore, we shall not investigate this in detail.

Seeing that services can also have non-functional requirements such as availability, accessibility, conformance to standards, integrity, reliability, scalability, security and transactionability, which are commonly expressed in terms of Quality of Service (QoS) (cf. Papazoglou and Ribbers, 2006,

p. 353), in this thesis, we focus only on mediating the computational aspect of services; that is, the functionalities that they provide.

Additionally, for the purpose of process mediation, we assume that systems who want to collaborate expose their functionalities as services (i.e. through Web services). These services are exposed publicly, i.e. they can be accessed by other external systems. Also, we assume that these services cannot be changed and are therefore fixed. This also implies that these services already exist. Finally, a service or services of one system do not have a direct functional relationship with those of the other system or systems it wishes to collaborate with; i.e., their services were not designed to match beforehand.

4.2. State-of-the-art service mediation approaches

This section describes some state-of-the-art service mediation approaches. For each approach, we first provide a brief overview, then a general description of the set of procedures it follows to solve the integration problem (i.e. its methodology or framework). We are particularly interested at how each approach involves business domain experts in the business-level design of service mediation solutions as we have described in Chapter 2. We compare these approaches throughout the section. Again, we focus on *process mediation* only. Naturally, other approaches that only support data mediation are not included. This survey extends the work of (Mantovaneli Pessoa, Quartel and Van Sinderen, 2008).

The following mediation solutions were derived in the context of the problem scenarios[11] presented by the Semantic Web Service (SWS) Challenge[12]. These scenarios involve two enterprises (Blue and Moon) each having their own existing systems exposing public services. These enterprises want to perform some business collaboration together but heterogeneity in data structures and protocol interactions prevent them from doing so effectively. Teams start with the same set of WSDL documents, sample XML messages based on some message structures described in XSD, and case descriptions as inputs to their approach. The challenge then is to ask participating academic and industry researchers to design a Mediator using their own methodologies and technologies to resolve such differences.

4.2.1. DERI approach

Overview

The Digital Enterprise Research Institute (DERI) team employs a semantic Web service framework based on the WSMO (Web Service Modeling Ontology) conceptual model, WSML (Web Service Modeling Language) language for service modeling, WSMX (Web Service Execution Environment) middleware system, and WSMT (Web Service Modeling Toolkit) modeling framework for solving application integration problems (Zaremba, Herold, Zaharia and Vitvar, 2008; Hasselwanter, Kotinurmi, Moran, Vitvar, Zaremba, 2006; Roman, et al., 2005).

WSMO elements are expressed in terms of *ontologies* which provide formal semantics to the information used by all other components (i.e. through concepts, relations, axioms, instances, etc.), *Web services* which represent behavioral aspects that must be semantically described in terms of non-functional, functional, and behavior properties, *goals* which specify objectives which a client might have when invoking a Web service, and *mediators* which provide interoperability among structural, semantic or conceptual mismatches between the components.

[11] http://sws-challenge.org/wiki/index.php/Scenarios
[12] http://sws-challenge.org/wiki/index.php/Main_Page

Methodology

The methodology is divided into two phases: integration *setup* phase and then the integration *runtime* phase. In the setup phase, message descriptions from the WSDLs of collaborating systems are semantically enriched using WSMO ontology. Mapping rules between ontologies of collaborating systems are at the same time described in WSML.

In order to connect existing systems using WSMX, adapters are used to resolve differences in their communication protocols and message structures. Since WSMX operates at the semantic level, adapters facilitate the "lifting" and "lowering" of XML messages to and from their equivalent WSML ontology using XSLT transformations; that is, adapters lift XML schema described in WSDLs to the WSML ontology, and is "lowered" in reverse. At the same time, adapters also handle the application logic of identifying goals to be sent to the WSMX environment.

Goals are used in WSMO to describe users' desires which represent how high-level objectives can be executed through Web services. They are achieved by selecting available Web services that can best satisfy them. Users can describe goals in terms of requested capability (desired functionality of the Web service) and choreography (how a user wants to interact with the environment). They allow services to be discovered, executed and mediated by the WSMX environment. WSMO goal specifications are written in WSML.

For the WSMO goal to be executed, an appropriate WSMO Web service has to be discovered. Like goals, WSMO Web services declare their capabilities (i.e. tasks they are able to accomplish) using its ontology. WSMO Web service capabilities are described in terms of pre-conditions, assumptions, post-conditions, and effects, and interaction behaviors which are all specified in WSML. To resolve heterogeneity between WSMO services, the WSMO Mediator overcomes mismatches at the data, protocol and process levels.

Choreographies of WSMO Web services are modeled as Abstract State Machines (ASM). ASMs allow the behavior of the Mediator to be abstractly described in terms of states and guarded transitions. States are described in terms of the WSMO ontology and transition rules dictate how changes between states are achieved.

Finally at runtime, the interactions between the collaborating systems are executed by the WSMX environment which facilitates the runtime integration process between systems. Figure 4-4 shows an overview of this approach.

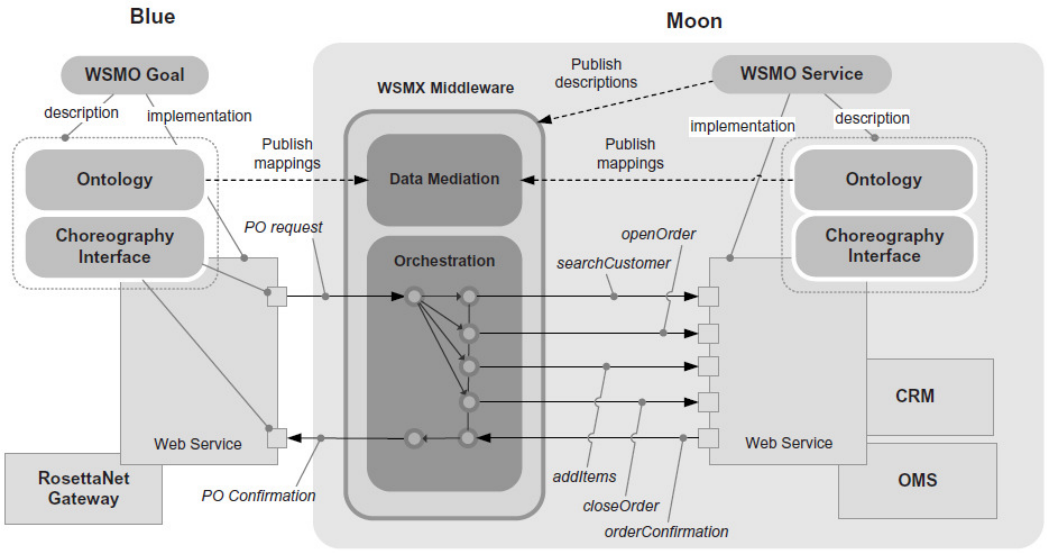


Figure 4-4: WSMO mediation architecture

Observations

We observe that the design of the service mediation solution using the WSMO framework focuses largely at a technical level. In particular, when users want to specify high-level objectives as WSMO goals, they need to do describe this using WSML. This could be quite challenging for business domain experts as they need to understand the language constructs and syntax of WSML before they can express their high-level objectives as WSML goals correctly. We thus find this to be too technical for business-level use.

Furthermore, the behavior design of the Mediator, expressed in WSMO choreographies, is based on Abstract State Machines. This may neither be intuitive nor easy to read, understand, or use for business domain experts since the behavior of the Mediator will have to be deduced from transition rules. Business domain experts may thus be expected to have some knowledge in ASM methodologies in order to read and understand how the behaviors are described. We consider this again to be too technical (Quartel, et al. 2008b)

4.2.2. SWE-ET approach

Overview

The SWE-ET approach for service mediation is proposed by the Politecnico di Milano and CEFRIEL team (Brambilla, Stefano, Della Valle, Facca and Tziviskou, 2008; Brambilla, Celino, Ceri, Cerizza, Della Valle and Facca, 2006). The framework incorporates Business Process Modeling Notation or BPMN (White, 2004) for specifying business processes, Web Modeling Language or WebML (Ceri, Fraternali and Matera, 2002) for designing and developing semantically rich Web applications, and WSMO for semi-automatic elicitation of semantic descriptions. The methodology puts emphasis on graphical modeling of multi-actor, asynchronous communication of dynamic cross-enterprise applications.

Methodology

The methodology begins through requirements specification by collecting and formalizing information about the application domain. Process design follows where the Mediator process specification is modeled manually at a high-level in BPMN. The process model formalizes the orchestration of the Web Services from collaborating systems, and defines the states pertaining to the mediation process.

The data design process follows where the resulting BPMN model is used to automatically generate WebML skeletons. Workflow information is then derived from the skeletons as hypertext and data models which are manually refined later to complete the design of the Mediator. The hypertext model is refined by designing the activities that transform functional requirements into one or more Web services and Web site views – a graph of pages that describes how a user can perform specific activities. These pages consist of units describing atomic information to be published which are related together though links describing the navigational paths from one unit to another while performing operations on the underlying data (e.g. create, read, update, delete). On the other hand, the WebML data model can also be refined in terms of E-R Diagram models. These models can then be used to automatically generate a running Web application.

Finally, the created artifacts (i.e. BPMN process model, data model, hypertext model, and WebML skeleton) are used to derive the WSMO specifications (i.e. goal, service, choreography, mediator) later deployed to the Semantic Execution Environment (i.e. WSMX) environment. The Mediator can now be invoked as a Web service by collaborating systems. Figure 4-5 provides an overview of the methodology.

Observations

The SWE-ET methodology is quite vague as to how business experts can participate in the design of the integration solution. We surmise that perhaps they participate in the design of the process models using BPMN. Business domain experts can use BMPN to read and understand business processes diagrams at a higher level, but additional implementation information must be supplemented by process implementers (BPMN-FAQ, 2004). Still, it will be quite difficult for business domain experts to participate in the design of the data and hypertext models. Data models are described using E-R Diagrams and requires some technical knowledge; on the other hand, they will also have to understand hypertext primitives (site views, pages, units, links, etc.) to model hypertext models to understand the chain of interactions between activities which are observed by some authors as less intuitive (Pessoa, et al, 2008).

Although the methodology supports requirements specification as its first activity, it is not clear how high-level goals expressed in the requirements are analyzed and structured to influence the design of the Mediator. The goal aspect indicated in the methodology is confined only as a WSMO specification, derived from BPMN model artifact, which is again too technical and may be difficult for non-technical business domain experts to grasp as we have described earlier under the DERI approach.

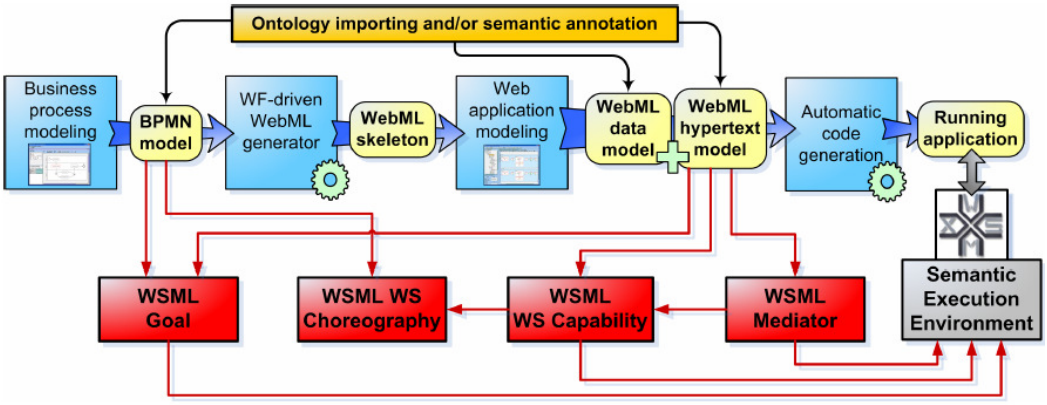


Figure 4-5: The SWE-ET approach for service mediation

4.2.3. jABC/jETI framework

Overview

The jABC (java Application Building Center) framework uses model driven application development based on Lightweight Process Coordination (LPC) (Steffen, Margaria, Nagel, Jorges and Kubczak, 2006; Kubczak, Margaria, Steffen and Nagel, 2008; Margaria, et al, 2007). It allows product developers and system/software designers to develop service-based applications by using reusable building-blocks into hierarchical (flow-) graph structures. A set of plug-ins provide additional functionalities to animate, analyze, simulate, verify, execute and compile jABC models.

LPC allows application experts, who are non-programmers to participate in the design of coordination models by using jABC in dragging and dropping predefined building blocks called SIBs (Service Independent Building Blocks). SIBs encapsulate functionalities used within an application or service. SIBs are used to create the workflow of the Mediator as a Service Logic Graph (SLG) which represents service composition models. SIB can contain a single functionality or may in fact be sub-graph (another SLG).

Methodology

The existing WSDLs from collaborating systems are first imported. SIBs are then automatically generated from WSDL descriptions which represent the operations described in the WSDL. The operations and messages described in the WSDL are then mapped as parameters to the SIB structure. The SIB structure provides a rich hierarchy of the messages imported from the WSDLs. The generated SIBs are then modeled manually to design the workflow as SLGs paying attention to the constraints that underlie the development of the business logic. The generated model can then be furthered analyzed, verified and simulated through a set of jABC plug-ins. The composite SLG, which may be composed of a hierarchy of sub-SLGs are then transformed into a single SIB which now represents the Mediator service – the GraphSB. From the GraphSB, an executable code is then generated and is deployed on a server using the Apache AXIS framework. A WSDL description that contains all the necessary information to access the deployed service is then generated as a Web service which can then be called by users. Figure 4-6 shows an overview of the methodology.

The jABC framework uses a set of plug-ins that performs various functions. Within the framework, WSDL service descriptions are imported and automatically transformed into SIB structures. Designing the SLG is done within the jABC framework by dragging and dropping SIBs into the editor. The generated choreography models are then checked using a core plug-in called GEAR which uses formalisms such as μ-calculus or temporal logic. The generated GraphSB is converted to code within the jABC framework (using CodeGenerator) and is executed using the Tracer plugin which serves as the interpreter (or virtual machine) for jABC service models. It also uses jETI facility to communicate with remote services such as those provided by the collaborating legacy systems.

Observations

The LPC-based framework of jABC is divided into two types of users working collaboratively: the *SIB Expert* and the *Application Expert*. SIB Experts are (Java) developers with detailed knowledge about the development of SIBs and appropriate plug-in interfaces. Application Experts are those who have detailed knowledge about the process or system under development but are not necessarily programmers with a technical background. The focus of jABC is on Application Experts and purports simplicity of use with its basic ideas easily conveyable to new participants in less than an hour.

The business logic of the application is modeled by Application Experts by manipulating SIBs that correspond to existing (or required) services. If more SIBs are needed, the Application Expert uses a SIBCreator Plugin to temporarily create a placeholder for the required SIB. Implementing the functionality of existing SIBs and creating missing ones are done in cooperation with the SIB expert. The also take care of the integration of legacy systems at the SIB level and the persistency layer (Steffen, et al., 2006).

We, however, observe that Application Experts are still relegated to a level of design space where they still need to be aware of the services available (as SIBs) and compose them to form SLGs. In turn, a necessary pre-requisite knowledge about service and service-oriented design principles (e.g. service cohesion, granularity, reuse, encapsulation, etc.) is required of them which still require some level of technical expertise. This is especially needed when, for example, workflows need to be refined to a better granularity as the methodology advocates. In the end, although they are not programmers, they still need to know service-oriented design principles.

Additionally, the framework does not also have explicit support/explanation as to how higher-level requirements, such as goals, can be structured and incorporated in their approach which may provide better involvement of business domain experts specifying requirements of the integration solution.

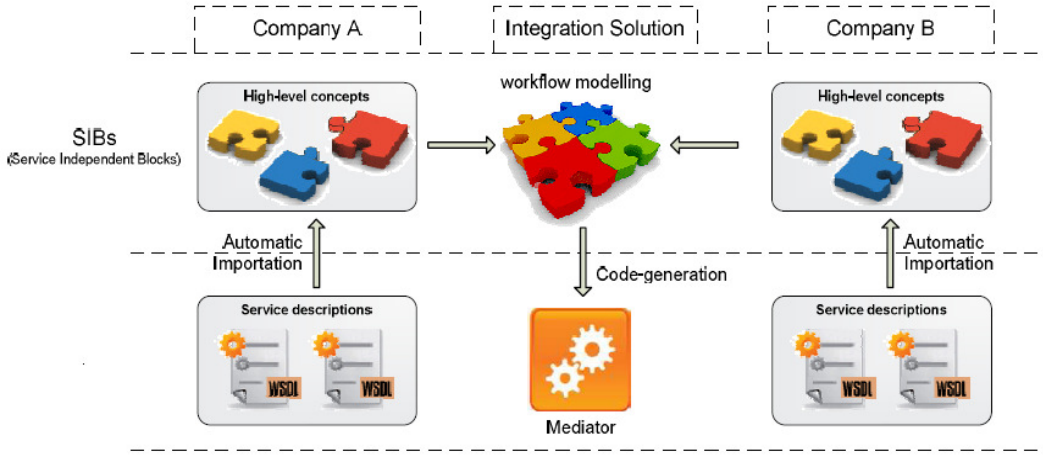


Figure 4-6: jABC/jETI framework for service mediation (adopted from Pessoa, et al, 2008)

4.2.4. COSMO approach

Overview

The COSMO (*CO*nceptual *S*ervice *MO*delling) is a service-oriented framework to model and reason about services, and to support service operations such as composition and discovery either at design or run-time. The University of Twente and Novay has applied the COSMO framework for service mediation with model-driven, service-oriented and semantic web techniques. The methodology is targeted at giving business domain experts the opportunity to participate actively in the design of the integration solution by providing a design space independent of any underlying technology using model-driven techniques (Quartel, et al., 2008a; Quartel, Pokraev, Dirgahayu, Mantovaneli Pessoa and Van Sinderen, 2008b).

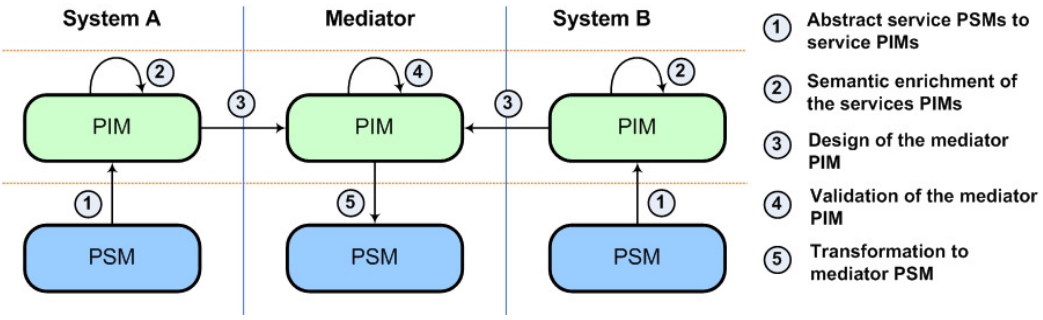


Figure 4-7: The COSMO approach for service mediation

Methodology

The method starts by "lifting" service description described in the WSDLs to a platform independent level which means, in Model Driven Architecture (MDA), transforming Platform Specific Models (PSM) to Platform Independent Models (PIM). Doing so avoids unnecessarily complicating the design space and thus providing more opportunity for business domain experts to be involved in the design. Business domain experts don't need to understand WSDLs to design the integration solution. The second step involves semantically enriching PIM information which cannot be automatically derived from the WSDL. This is done because WSDLs do not inherently provide interaction protocols (i.e. how the sequence of message execution is specified)

so that this information is supplemented through some text documentation in natural language, stakeholder interviews, or even code inspection. This enrichment is done so that the PIM is designed completely and precisely allowing better reasoning and generation of the mediation solution later on. The third step involves the actual design of the Mediator at the PIM level. This usually involves splitting the integration solution in two areas which may be done in parallel: generating the *behavior* and *information* models. The information model unifies the differences in the data representations and interpretations between systems while the behavior model composes requested and provided mismatching service by relating their operations (i.e. matching the input of an operation call to the output of another and their constraints). The fourth step involves validation of the composed service Mediator using techniques such as interoperability assessment (Quartel and Van Sinderen, 2007b) and simulation of generated behavior and information models. The final step includes transforming the designed PIM models of Mediator to PSM as its technology implementation. Figure 4-7 shows a diagram of the methodology.

In deriving the behavior (operation) and information (message) information contained in the WSDL files at the PSM level, the methodology uses Interaction Systems Design Language (ISDL) to model behaviors and UML to model information at the PIM level. The Grizzle tool, an integrated editing and simulation environment for ISDL, provides an import functionality that transforms operations described in the WSDL files as operation calls in the ISDL model. The information model is represented in UML by first generating the Java classes of the XSD message types contained in the WSDL using JAXB or JAX-WS. The design of the behavior model in the third step is done entirely in ISDL using Grizzle. To express data transformations, the methodology uses a Domain-Specific Language (DSL) to define mappings as functions between two data objects and bind them together at runtime. Once the ISDL model is defined, validation proceeds by using Sizzle (also within the Grizzle environment) to simulate operation calls and data transformation at the PIM level. Finally, the validated ISDL model can then be transformed into Business Process Execution Language (BPEL)[13] as the PSM implementation.

Observations

The COSMO approach for service mediation purports to involve business domain experts in the design of the integration solution by raising the design environment to a level that does not require them to understand technology specification such as WSDL and BPEL. While supporting MDA, the methodology only covers design abstraction at the PIM level. This means that business domain experts will need to read, write and understand the behavior model which is described in ISDL. They therefore need to be familiar with its constructs. Apart from this, when designing the service composition, they are also expected to at least have some knowledge of service modeling, composition and refinement principles advocated by the COSMO framework.

It is, however, interesting to see how the methodology fairs when extended to support the CIM level where requirements are described without regard to how they are structured and implemented in any platform thus bringing the design space closer to the level comprehensible by business domain experts.

4.2.5. Discussions

Expanding and summarizing our observations, we note that:

- There is currently a strong research interest in the area of service mediation. Practitioners and researchers recognize that problems brought about by process and data mismatches between heterogeneous systems need to be addressed in order to provide better solutions for enterprises who want to innovate by making their IT systems more responsive to volatile business demands.

[13] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

- This strong research interest is manifested through the number of approaches that use varying technologies and techniques as reported in literature. All of these approaches put strong focus on enabling service-based integration using Semantic Web technologies.
- Even though the SWS Challenge problem scenarios present rigorous requirements for service mediation, the various approaches have shown that current technologies are sufficient to meet the requirements.
- Model Driven Architecture (MDA) is central to all these approaches. The approaches involve the basic idea of raising the design space to the semantic level where interactions are analyzed more abstractly. The design of the process and data mismatch solutions no longer need to be handled at the platform specific level (e.g. by direct manipulation of WSDL, XSD files or application codes).
- While most approaches have raised the design abstraction to a level that is not platform specific, we observe that most of them use technologies and techniques that are still too technically-oriented, and thus lean more closely to the technology side. For example, the DERI approach requires designers to be familiar with WSML in specifying process and data interactions including the heavyweight architecture of WSMX framework. The COSMO team requires business domain experts to understand ISDL to model the behavior of the Mediator service.
- Although most of the approaches support the importance of requirements specification, they do not describe how these requirements are identified and refined in a structured manner so as to guide the composition of the Mediator service.
- Finally, there is still a need to bring the design of integration solutions closer to business domain experts. Interestingly, these approaches provide a promising opportunity for such improvement. We thus propose business-driven approaches that capture and structure requirements to be made an integral part in the design of service mediation solutions. It is in this area where our research contributes to.

4.3. Chapter summary

This chapter is largely divided into two parts: first, we provide definitions about service mediation as used in this thesis; second, we assess current service mediation approaches as to how they involve business domain experts in the design of their integration solutions.

Today's autonomous enterprise systems often find it difficulty to interoperate (i.e. exchange and use each other's information effectively) if they were originally designed to exchange information with external systems (i.e. they follow monolithic architectural paradigms), they follow different industry standards, or they use different hardware and software platforms.

More specifically, a process interoperability mismatch occurs when systems use services that define different messages or different ordering of message exchanges. Such a mismatch happens when systems want to collaborate but have publicly exposed services that are fixed and are difficult, costly, or if not impossible to change for the purpose of integration. A data interoperability mismatch occurs when systems use different information models (or vocabularies) to describe the messages that are exchanged by their services. A Mediator which is a software component that sits between collaborating systems is therefore need to handle process and data interoperability problems between one or more enterprise systems through their exposed services.

Surveying current service mediation approaches, we find two salient observations: Firstly, although such approaches follow model-driven techniques, the technologies and methodologies they use are limited only to the PIM level which essentially means that business domain experts,

whom we consider as those without sufficient technical background, still need to be familiar with the technologies that were used to design the Mediator. The approaches we have surveyed therefore are to a large extent still very much technically-oriented. Lastly, although it is quite obvious that the approaches use the case descriptions of the SWS Challenge as the starting point to draw the requirements from, it is unclear how the approaches structure the requirements and translate them into technical implementations. Treatment of the requirements as design artifacts is not very clear.

Part II

Solution Design

This part is comprised of Chapters 5 and 6. The goal of this part is to present a solution that combines goal-driven and service mediation approaches to allow better participation of business domain experts in the design of integration solutions. We present the solution in Chapter 5. We supplement the solution with software tools from the industry, open source community, and academe and describe how such tools help realize the solution. We present the tools in Chapter 6.

5

A goal-driven service mediation solution

This chapter presents in detail the mediation solution we propose in this research. We describe how goal-driven and service mediation approaches can be combined to provide a better opportunity for business domain experts to be actively involved during the design of the integration solution. We show how model-driven techniques can bind these approaches together.

The structure is as follows: Section 5.1 first provides an overview of the paradigms that form the basis our service mediation solution; namely, our chosen goal-driven, service mediation, and model-driven approaches. Section 5.2 briefly describes the technologies we have chosen to translate goals into different models in the MDA stack. Section 5.3 describes the resulting generic architecture of our solution. Section 5.4 details the methodology step by step. Section 5.5 ends this chapter with a summary.

5.1. Overview of approaches

The *solution* proposed in this thesis is born of three approaches: we bridge the gap between *goal-driven* and *service mediation* approaches for business-level, service-oriented design of interoperating enterprise systems using *model-driven* techniques. By "solution", we mean an overall approach that entails both a *methodology* and an *architecture*.

Before we proceed into describing the details of the solution, we first discuss these approaches, and thereafter identify areas where they can be combined. We base our solution on the following paradigms:

- *ARMOR + ArchiMate for goal-driven Enterprise Architecture design*

  We involve business domain experts in the design of integration solution by providing them concepts and tools to state their requirements. Assisted by business analysts, they do this using goal modeling concepts provided by ARMOR. As we have shown in Chapter 3, Goal Oriented Requirements Engineering captures the motivations behind the design of software systems and guides business domains experts in the process. We argue that business domain experts are at the best position to describe the requirements of the service mediation

solution through goals as it provides them a sufficient level of abstraction in specifying and validating system design choices at the business level, and for communicating such choices among different stakeholders.

Our choice of ARMOR as the goal the modeling language is largely motivated, firstly, by the nature of its design: namely, suitability for the design of enterprise architectures, ability to document, communicate and reasons about requirements, and provisions for an easy-to-use set of language constructs. Secondly, since ARMOR adds requirements engineering capabilities to ArchiMate, a modeling framework to the design of Enterprise Architectures, we are able to take advantage of the features of the latter which are mainly towards better business-IT alignment and requirements traceability. Thirdly, as ArchiMate provides an integrated modeling approach to various domains of the enterprise, various stakeholders (i.e. the business domain experts, business analysts and IT experts) are able to understand how requirements, modeled as goals, are eventually realized as or supported by architectural artifacts at the business, application and technology layers of the Enterprise Architecture. We find this an important feature to the design of the service mediation architecture.

- *COSMO approach for service mediation*

  The COSMO framework for service mediation uses model-driven, service-oriented and semantic web techniques. The framework proposes a methodology that is currently targeted at giving business domain experts the opportunity to participate actively in the design of the integration solution by providing a design space independent of any underlying technology (e.g. WSDL, BPEL, ISDL) using model-driven techniques.

  However, while supporting MDA, the methodology only covers design abstraction at the PIM level. We observe that used PIM specifications (i.e. ISDL, Java) are still too technical for non-technical business people to fully appreciate. Business domain experts will need to read, write and understand the behavior model which is described in ISDL. They therefore need to be familiar with its constructs. Apart from this, when designing the service composition, they are also expected to at least have some knowledge of service modeling, composition and refinement principles advocated by the COSMO framework.

  We propose an improvement to this situation by incorporating goal-driven approaches. Using ARMOR, we specify how the composition will be achieved by first modeling the goals at the CIM level through goal analysis/identification, decomposition/refinement, or abstraction techniques as described in Section 3.4 – essentially extending the framework to an abstraction more suited for business-level analysis of requirements. We then use ArchiMate to model how these goals are implemented and realized by the underlying Enterprise Architecture.

- *Iacob, et al. (2009) framework for goal- and model-driven design of SOA*

  Once we have the requirements of the service mediation solution expressed in terms of goals driven by business requirements, we further investigate the use of *model-driven techniques* to transform these abstract goals into technology-specific implementations, and investigate how business rules can be used in the process. For this purpose, we use the framework proposed by Iacob, et al. (2009) for goal-driven design of service-oriented systems using model-driven techniques. The framework seeks to incorporate a business view in SOA development where high-level goals are refined and operationalized as business rules. These rules are then transformed into a language where it can be executed and eventually incorporated into the design and composition of services. MDA is the central binding concept that allows goals to be transformed into business rules, to services, and finally to executable rule expressions

  The framework is mainly aimed at designing service-oriented software systems in *general*. This research explores the possibility of using this framework further in the context of service mediation. The framework is well suited for our requirement, which is to allow better participation of business domain experts in the design of an integration solution, as it uses

goals as a central element. Furthermore, we investigate how business rules can be specified so that they can be integrated in the composition design of services for service mediation. Lastly, with MDA as one of the key elements of the framework, we investigate how it can be combined with the MDA-based service mediation approaches of COSMO.

5.1.1. ARMOR + ArchiMate

5.1.1.1. ARMOR

We have briefly described ARMOR in Section 3.6.4. This section provides an overview of its abstract and concrete syntaxes important to the understanding of our solution. We take the descriptions from Mantovaneli Pessoa, Van Sinderen and Quartel (2009), and Quartel (2009a).

Abstract syntax

Figure 5-1 shows the abstract syntax of ARMOR. A *goal* is defined as some desired result that is to be realized in the problem domain. A goal may need to be refined into hard and/or soft goals. With *hard goals*, the criteria for satisfying a goal are clear and precise; whereas, the criteria to satisfy a *soft goal* are typically vague or subject to interpretation.

A *requirement* is a goal that can be assigned to some system which is made responsible for the satisfaction of the goal.

A *business rule* describes how an enterprise should conduct its business. It may constrain or influence some behavior, object or structural element. It can also be considered as a type or an implementation of a requirement; and hence, it can be considered as a refinement of a goal as well. They are defined in a declarative way, are actionable (i.e. a person can assess whether or not the enterprise complies with the business rule), and are executable (e.g. by some rule based engine).

A *use case* describes multiple or alternative sequences (also called scenarios) of interactions to satisfy goals. A use case may have a *general* relation which means that the child use cases contain all interactions of the parent use case, *include* relation which means that the sequence of interactions of the child use cases are included in the parent use case resembling the notion of a subroutine, and an *extend* relation which means that the parent uses case is extended with alternative scenarios.

Concrete syntax

Figure 5-2 shows the concrete syntax of ARMOR. The constructs described earlier have their own modeling constructs in ARMOR. We now describe the other relation constructs. When goals are refined into a set of sub-goals, several refinement constructs can be used:

- `AND`-realization (conjunction) means that one or more *all* sub-goals must be satisfied to satisfy the parent goal,
- `OR`-realization (disjunction) means that *at least one* or more alternative sub-goals must be satisfied to satisfy the parent goal

Conflict or contribution relations are also used to describe the relationship between goals. Conflict relations are used when a hard goal conflicts the satisfaction of another hard goal. Contribution relations describe how either a soft or hard goal can contribute to the satisfaction of another soft or hard goal. Properties of the contribute relation are:

- `type`: the type of contribution (i.e. positive or negative), and
- `strength`: the strength of the contribution (i.e. weak or strong)

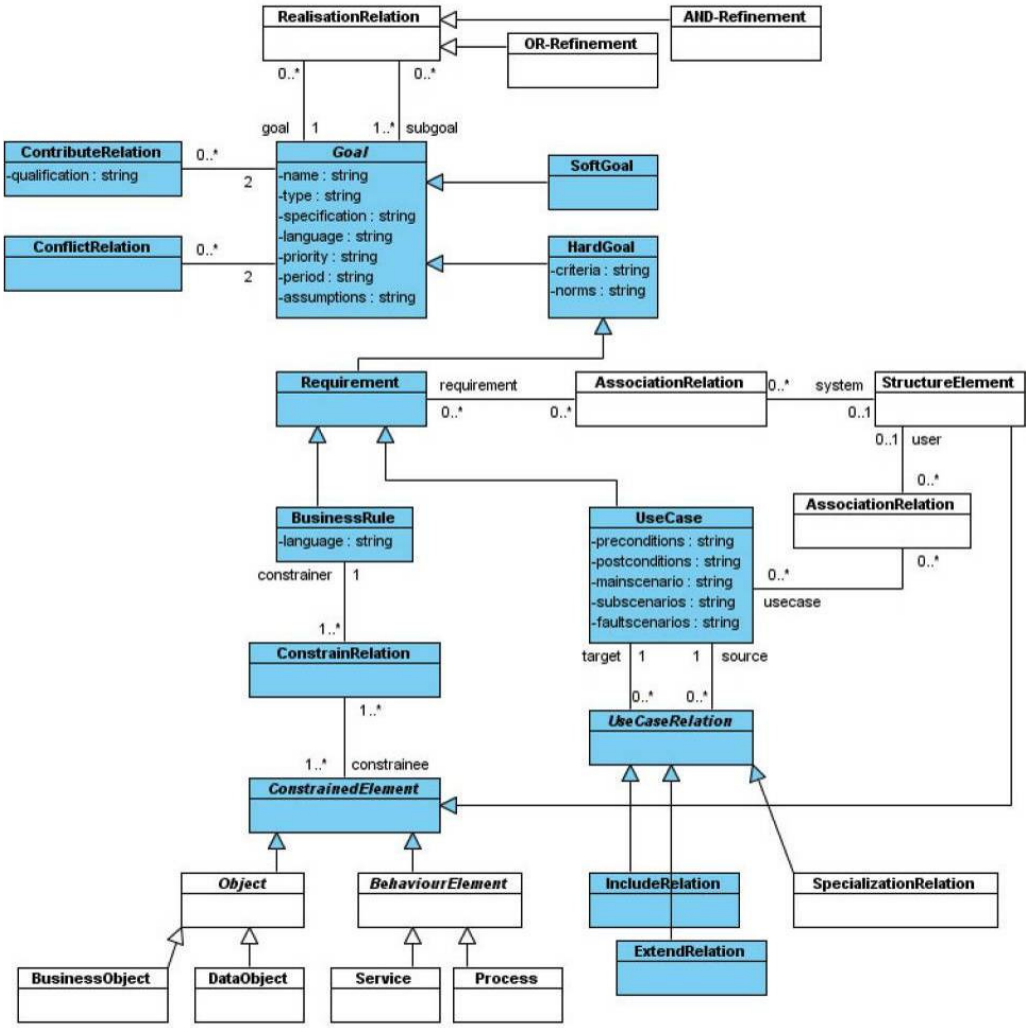


Figure 5-1: Abstract syntax of ARMOR (Quartel, 2009a)

Abstract element	Concrete notation
Hard goal	Hard goal
Soft goal	Soft goal
Requirement	Requirement
Use case	Use case
Business rule	Business rule
Realisation	
AND realisation	
OR realisation	
Contribute relation	+/-
Conflict relation	
Include relation	<<include>>
Extend relation	<<extend>>
Constrain relation	<<constrain>>

Figure 5-2: Concrete syntax of ARMOR (Quartel, 2009a)

5.1.1.2. ArchiMate

ArchiMate (Lankhorst, et al., 2005) is a modeling language for Enterprise Architectures. It is high-level design language for Enterprise Architecture that is accompanied by a set of techniques and guidelines for modeling, visualization, and analysis of architectures. By providing a unified way of modeling Enterprise Architectures, it intends to eliminate ambiguities and confusion that arise when various domain architectures of an organization like business process, application, information, and technical architectures come together. It uses a set of concepts within and relationships between architecture domains, and offers a simple and uniform structure for describing the contents of these domains. ArchiMate believes that when various domain architectures are integrated together, organizations are able to better align business and IT operations with its strategy allowing them to quickly respond to changes in the business environment.

A *service* is a key concept in ArchiMate as it provides relationships between domains. It leads to a service orientation where Enterprise Architecture models are layered into different views. The service concept links these different layers; that is, services made available at higher layers are realized and implemented in lower layers. ArchiMate distinguishes three main layers:

- The *Business* layer where products and services to external customers are offered, which are realized in the organization by business processes performed by business actors and roles.
- The *Application* layer supports the business layer with application services which are realized by (software) application components.
- The *Technology* layer offers infrastructural services (e.g., processing, storage and communication services) needed to run applications, realized by computer and communication hardware and system software.

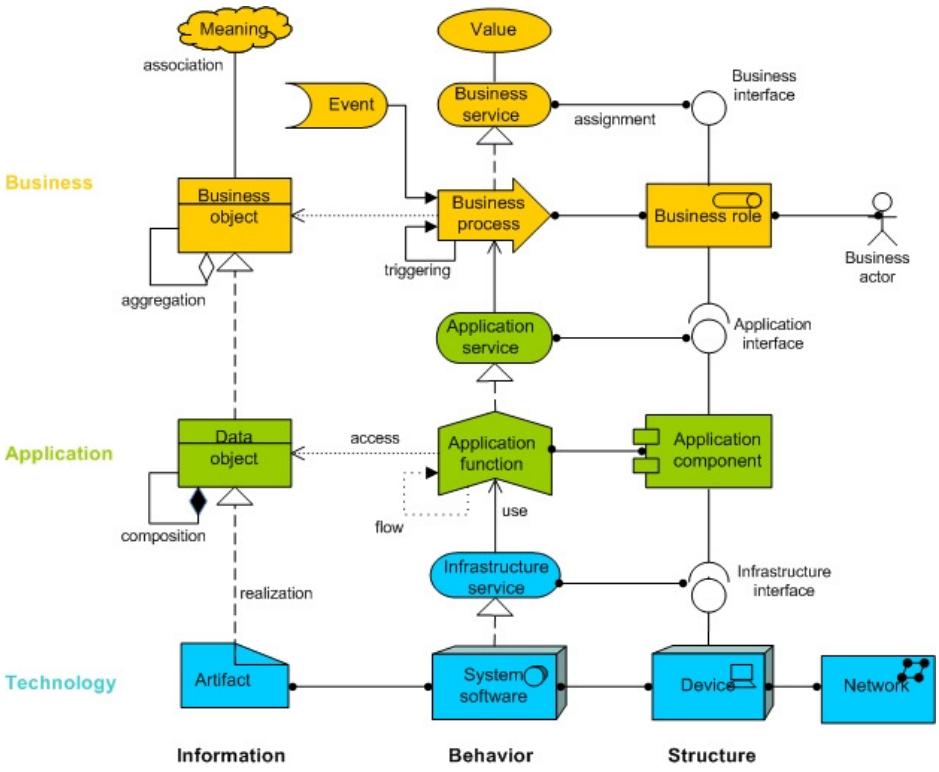


Figure 5-3: Basic concepts of ArchiMate[14]

[14] Adopted from http://www.archimate.org/en/about_archimate/what_is_archimate.html

5.1.2. The COSMO approach to service mediation

COSMO (Quartel, Steen, Pokraev and Van Sinderen, 2007a) provides a framework to model and reason about services, and to support service operations such as composition and discovery either at design or run-time as shown in Figure 5-4.

A service is an established effect as a result of interactions between systems. It can have several aspects: *information, behavior, structure* and *quality* representing categories of service properties that need to be modeled. Modeling systems that provide or use services require knowledge of their interconnection structure (e.g. ports and interfaces). Knowledge of activities that systems perform, including their relations, constitute the behavior aspect. Information that is managed and exchanged among systems also needs to be modeled. Modeling non-functional characteristics (e.g. cost associated with response time) of services concerns the quality aspect.

The framework also distinguishes modeling three generic abstraction levels: *goal*, *choreography* and *orchestration*. Goal level modeling describes a service as a single interaction, where the interaction result represents the effect of the service as a whole. Choreography level modeling refines these goals describing the service as a set of multiple related, more concrete interactions. Orchestration level modeling describes the implementation of the service using a central coordinator that invokes and adds value to one or more other services.

Finally, different roles of the systems may be involved in service modeling: the *user*, *provider* and *integrated* role. The integrated role abstracts from the distinction between a user and provider by considering interactions as joint actions, thereby focusing on what the user and provider have in common.

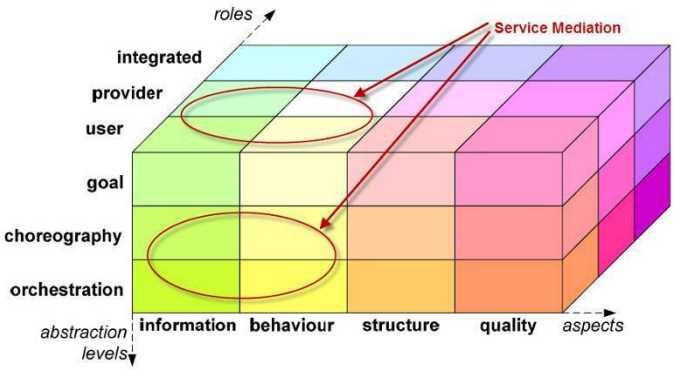


Figure 5-4: The COSMO approach

The COSMO approach to service mediation takes the design of the Mediator as a service *composition* problem. It mainly considers *choreographies* and *orchestrations* from the *behavior* and *information* aspect further distinguishing between *user* and *provider* roles. Furthermore, it uses MDA to provide a solution for semantic service-oriented mediation (Quartel, et al., 2008a; Quartel, et al., 2008b). The approach was briefly described in Section 4.2.4; we show again the figure in Figure 5-5.

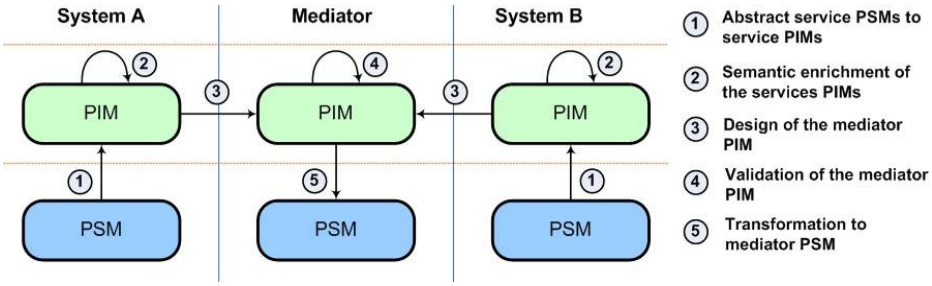


Figure 5-5: The COSMO approach for service mediation

5.1.3. Goal- and model-driven approach to SOA design

Driven by the need to enable organizations to have their IT systems react swiftly and coherently to the ever dynamic nature of business demands, Iacob, et al. (2009), Iacob and Jonkers (2008) propose a framework for *goal-driven design of service-oriented systems using model-driven techniques*.

The framework seeks to incorporate a business view in SOA development by using the concept of goals in eliciting, structuring, and modeling requirements. Goal models are used to capture the requirements at a level where it is easily understandable and verifiable by business domain experts. High level goals are further refined and operationalized as business rules. These rules are then transformed into a language where it can be executed and eventually incorporated into the design and composition of services. MDA is the central binding concept that allows goals to be transformed into business rules, to services, and finally to application code.

The framework *decouples* business rules from the business processes that use them; that is, the rules are treated as separate design or implementation artifacts. Should there be changes to the business rules such as the introduction of new laws, regulations, or an overall change to the business strategy of the organizations, service systems that implement them can respond to the required change rapidly – thus providing better business process and software agility. Also, as rules are decoupled, rule reuse is therefore possible.

Furthermore, the separation of business rules from processes also allows for better *transparency*; that is, business rules are no longer hidden or hard coded in business processes or even application code. Separating the business rules also allow organizations to manage them explicitly as, for example, when some form of rule repository is used to contain and manage all rules used by their systems.

The framework, shown in Figure 5-6, is divided into two *spaces*: *Design* and *G&BR* (Goals and Business Rule) vertically. They are divided horizontally into several layers of the MDA stack (i.e. CIM, PIM, and PSM). The Design space expresses models in design languages such as UML, business process modeling or architectural description languages. The G&BR space models goals and rules in special-purpose specification languages such as PRR, OCL, RuleML, etc.

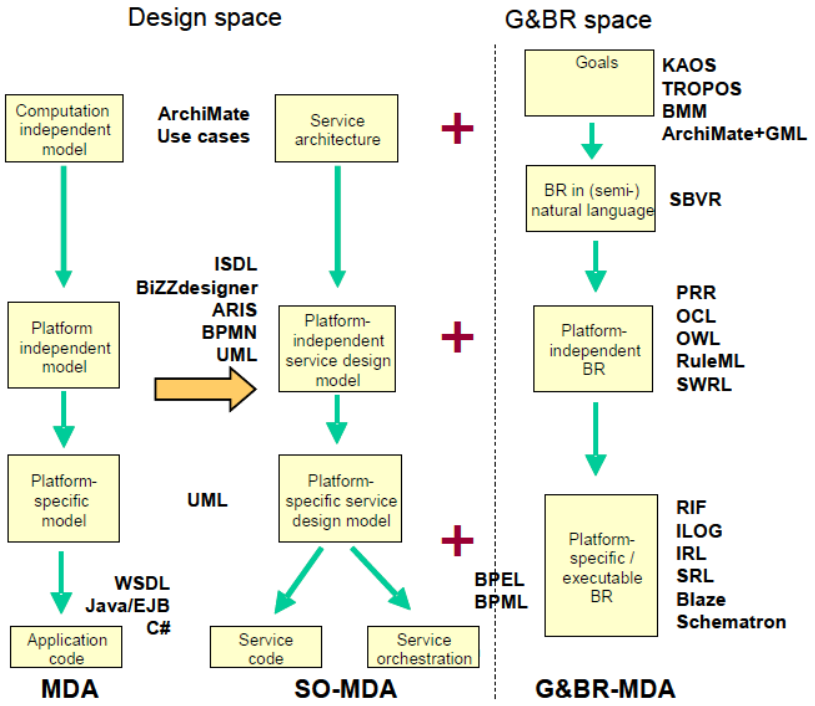


Figure 5-6: A goal-based, model-driven approach for SOA design (Iacob, et al., 2009)

The framework argues that it is possible to combine business rules in the model-driven design of SOA (indicated by the + symbol), where, in the G&BR space, they are mapped into the layers of the MDA (Design space). Additionally, the framework suggests that there is a strong symmetry between these spaces in that for any design model, there may be a corresponding rule set specification.

Goals, specified at a higher level, can be refined by first operationalizing them into more concrete business rules specified in near natural languages at the CIM level (e.g. SBVR). At the PIM level, these business rules are translated into a XML-based rule specification PSM levels for interoperability between other models. Finally, at the PIM level, they are eventually implemented as executable rules exposed as Web services. Aside from this vertical model-to-model transformation both form the Design and G&BR spaces, horizontal model merging transformation between the two spaces at different layers of the MDA stack is also possible. These vertical and horizontal transformations are currently the subject of further research.

5.2. Model transformation architecture

This section provides an overview of the *technologies* that we have used to realize our solution – basically, an instance of the Iacob, et al. (2009) framework described Section 5.1.3. Although Iacob, et al. (2009) argues that it is possible to combine G&BR and Design spaces correspondingly at the different layers of the MDA, we have been able to show this *directly* at the CIM and PSM layers and *indirectly* at the PIM layer as we shall explain later. Figure 5-7 shows an overview of these technologies positioned in the MDA stack and the Design and G&BR spaces.

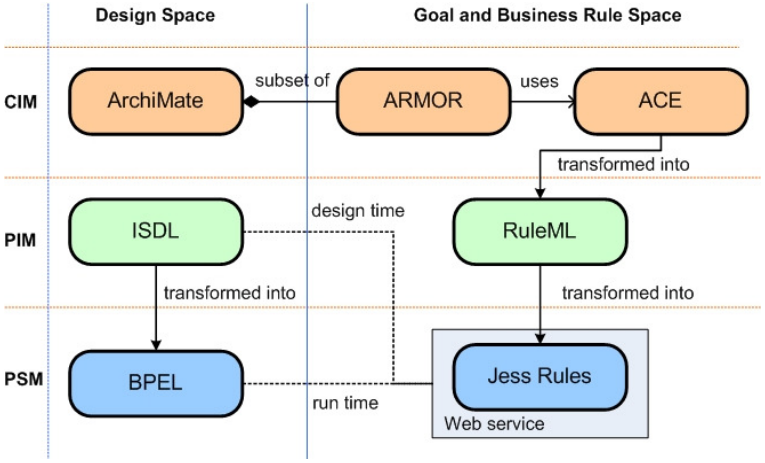


Figure 5-7: Model transformation architecture

At the CIM level, we use ARMOR to model goals and their refinement and operationalization into business rules. ARMOR is not currently designed to allow automatic transformations of business rules into controlled language specifications (e.g. Semantics of Business Vocabulary and Business Rules or SBVR (Object Management Group, 2008a)). Our solution to this has been to *manually* translate the business rules from ARMOR into the controlled languages of our choice – Attempto Controlled English (ACE).

At the PIM level, we use Rule Markup Language (RuleML), an XML-based rule language, to give an interoperable representation of ACE rules. In the Design space, behavior models are specified using Interaction System Design Language (ISDL). Currently, however, we find no studies that seek to represent rule-based languages such as RuleML into an equivalent representation in ISDL. Since ISDL speaks the language of services, we solve this indirectly by first encapsulating rules as Web services, and then combine them through their service specifications in ISDL.

Finally, at the PSM level Design space, the ISDL model is transformed into an executable business process specification such as the Business Process Execution Language (BPEL). To execute rules in the G&BR space, we transform the non-executable RuleML representation into Java Expert System Shell (Jess) rule expression – a rule-based scripting language and execution engine. We wrap the Jess rules as Web services using Java so that they can be combined at design time in ISDL at the PIM level, and invoked at runtime in BPEL at the PSM level. This research does *not* investigate transforming ArchiMate behavior constructs into ISDL.

Mapping the technologies at the MDA stack

ACE, being a controlled language, allows us to state business rules in near-natural English. Stating rules in near natural language allows business domain experts a better opportunity to validate their business requirements as it is written in a form that they can intuitively and naturally understand. This is a powerful feature that we would like to take advantage of. ACE sentences, however, face the problem of interoperability with other technologies if they are to be translated directly (Fuchs, Kaljurand and Schneider, 2006). For this, the authors of ACE have used Discourse Representation Structures (DRS) – a syntactical variant of first-order logic that eliminates ambiguities in natural language – as an inter-lingua for transformation to other languages (Fuchs, Hofler, Kaljurand, Schneider and Schwertel, 2005). ACE sentences have been transformed to other technologies, for example, to OWL-DL through DRS (Kaljurand and Fuchs, 2006).

To provide interoperability to our rules, and ensure that our solution is designed for change, we transform the ACE representation of business rules into and XML-based specification. We choose RuleML for this purpose. XML-based representations are, however, not naturally executable (aside from the fact that XML is not naturally understandable by business domain experts).

To provide an executable platform to our rules, we use Java Expert System Shell (Jess). Jess rules, by themselves, are naturally not interoperable with other rule-based technologies (in fact, JessML[15] – an XML-based rule language for the Jess has been created for the purpose of interoperability with, for example, RuleML). Jess is obviously not suitable for the understanding of business domain experts.

Why choose ACE, RuleML, and Jess?

Our choice of the aforementioned technologies has largely been based on their maturity, availability of transformations as reported in literature, and tool support. ACE has been around since 1995, and work continues until today. ACE is one of the most research controlled natural language, and there are is number of work describing its application.

RuleML, which started in 2000, is one of the early XML-based rule languages proposed in literature. New rule-based technologies have taken RuleML as base specification (examples are Reaction RuleML[16], and REWERSE RuleML or R2ML[17]).

Early work on Jess started in 1995, and it has, from then on, enjoyed a wide acceptance in the industry and academic community as evidenced by the large number of work relating to its application (Friedman-Hill, 2003). We choose Jess because of its strong connection with the Java programming language which allows us greater control and flexibility in the design of our rule-based solution. Other proprietary rule technologies such as IBM ILog[18] and Oracle Business

[15] http://www.jessrules.com/jess/docs/70/xml.html
[16] http://ibis.in.tum.de/research/ReactionRuleML/
[17] http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML
[18] http://www.ilog.com/

Rules[19] tend to hide implementation details from the user. We note, however, that the Jess rule engine is not open source or a freeware.

In the subsections that follow, we provide a brief overview of the technologies not previously discussed. We limit their descriptions only to the key concepts which we to deem important for the reader to better understand the details of our methodology.

5.2.1. ISDL

The *Interaction System Design Language* (ISDL)[20] supports the design of distributed systems by providing generic design concepts and a notation to model their structure and behavior at a higher level of abstraction. ISDL concepts have been adopted by the COSMO framework for modeling behavior interaction of services. Thus, ISDL is also used as a language to model the behavior of services in the COSMO framework. ISDL has been applied in other various systems, among them business processes, distributed applications and communication protocols.

This section provides a very brief overview of ISDL language constructs. For a full treatment of this topic, please see Vissers, Ferreira Pires, Quartel, Van Sinderen (2007), Quartel, Dijkman, Van Sinderen (2004), and Quartel, Ferreira Pires, Van Sinderen (2002), and Dirgahayu (2005).

ISDL models distributed systems architecture in three parts:

- The *entity model* which identifies system parts and their interconnection, representing the structure of the system.
- The *behavior model* which represents the functionality of the system parts and their interactions
- The *assignment relation* which defines the relationship between the entity and behavior models.

ISDL models can be expressed in a textual notation or a graphical notation. Figure 5-8 shows the graphical notation of the models.

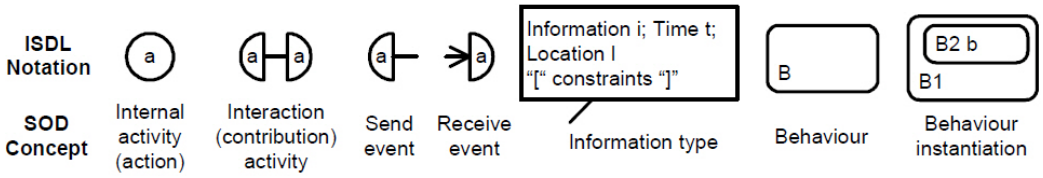


Figure 5-8: Representing service-oriented design concepts in ISDL (Quartel, et al., 2004)

An *action* (represented as a circle or eclipse) is a concept to model the successful completion of an activity performed by a single entity. An action is *atomic*; that is, it is an indivisible unit of activity at the abstraction level where it is defined. Consequently, an action either occurs or does not occur.

An *interaction* models the cooperation of one or more systems to produce a common successful result. The interaction concept describes possible results that can be produced, not how this result is achieved. An interaction can be considered as atomic when the same result occurs and is established for all involved systems; otherwise, they do not happen at all, and thus no result is established. An interaction occurs only when systems are willing to contribute to the interaction. The contribution of each system is modeled as an *interaction contribution*.

An *information type*, attached to an activity, represents the type of result of an activity, the time moment the result was established, and the location where result is available.

[19] http://www.oracle.com/appserver/rules.html
[20] http://isdl.ctit.utwente.nl/index.htm

A *behavior* (represented as a rounded rectangle) models the collection of activities and their relations. Each activity can be performed by the system alone or in cooperation with other systems. A *behavior type* is a definition of a behavior, while a *behavior instantiation* creates an instance of a behavior type. For example, in Figure 5-8, behavior `B2` instantiates behavior `B1`; thus, an instance of `B2` contains an instance of `B1`, called `b`. Behavior instantiation allows multiple instances of the same behavior to be created, and hence reused. A *behavior recursion* occurs when a behavior type instantiates a copy of itself.

A *causality relation* represents the relations between activities. It defines an activity's causality condition that must be satisfied to enable the activity to occur. Causality conditions are defined in three elementary conditions shown in Figure 5-9: (i) the *start condition* states that activity `a` is enabled at the beginning of some behavior independent of other activities, (ii) *enabling condition* `b` represents that activity `b` must occur first before `a` can occur, and (iii) *disabling condition* `¬b` represents that activity `b` must not occur first (nor simultaneously with `a`) to allow `a` to occur. `AND` and `OR` operators can be used to combine these elementary conditions. Figure 5-10 shows some simple examples.

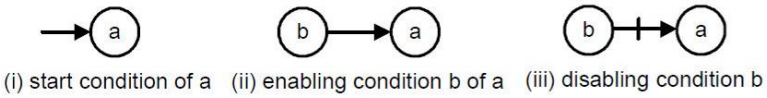


Figure 5-9: Basic action relations

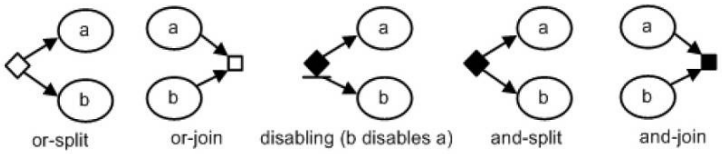


Figure 5-10: Causality relations combination

ISDL also supports the *operation* concept in COSMO (an improvement to the *send* and *receive events* in Figure 5-8). An operation represents an instance of a message passing composition. The operation concepts allows us to distinguish between *user* and *provider* roles in the design of orchestration and choreographies between behavior and information aspects of the COSMO framework (cf. Section 5.1.2) (Quartel, et al., 2008a).

In an *operation call,* sending a request is represented by an *invoke* while the receipt thereof is represented by an *accept*. Operation calls are usually assumed by system with a user role. In an *operation execution*, a receipt of a request is represented by an *accept* while the response to that request is represented as a *reply*. Operation executions are usually assumed by systems with a provider role. Optionally, an operation call may have *catch* component while an operation execution may have a *fail* component. This is usually omitted for one-way operations. Figure 5-11 shows how the operation concept is graphically represented in ISDL.

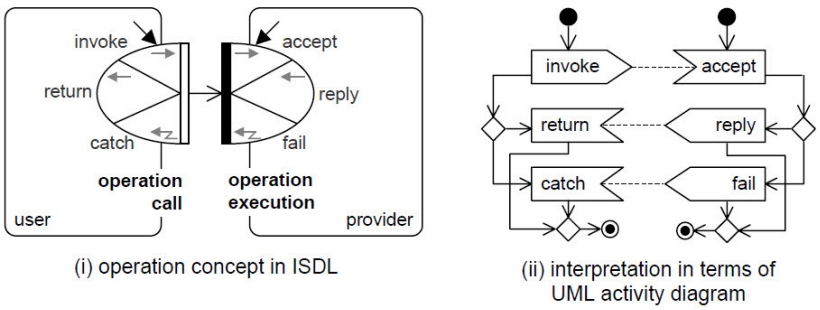


Figure 5-11: Operation concept in ISDL

5.2.2. ACE

Attempto Controlled English[21] (ACE) (Fuchs, Schwertel and Schwitter, 1999) is part of the Attempto Project developed by the University of Zurich's Department of Informatics and the Institute of Computational Linguistics which began in 1995.

ACE is a type of *controlled natural language* (CNL). CNLs are a subset of English (although other languages are also used) whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity (Schwitter, 2009). Two basic purposes of CNLs are to either improve human readability or facilitate computational processing of natural language. Until recently, controlled languages have been applied in area of knowledge representation in the semantic Web. For a thorough discussion of controlled languages, we refer you to Schwitter (2009).

The objectives of ACE are (cf. Fuch, et al., 1999): (i) to support the writing of precise specifications, (ii) to reduce ambiguity and vagueness inherent in full natural language, (iii) to encourage domain specialists to deliberately choose a clear and unambiguous writing style so that readers of a specification understand it in the same way as the writer, (iv) to make specifications computer processable, (v) to render specifications unambiguously translatable into formal specification languages, particularly into first-order logic, and (vi) to make specifications executable.

The following is only a brief treatment of the ACE language. For a full treatment of the language, consult Fuch, et al. (1999). ACE has *construction rules* that describe allowed sentence structures. It has *interpretation rules* that control the meaning of ACE text and resolve their ambiguities. Finally, it has its own set of *vocabulary* which is comprised of the following:

- *predefined functions words* which are used to establish the syntactic or logical structure of sentences (e.g. determiners (*the*, *an*, *a*), conjunctions (*and*, *or*, *nor*, etc), prepositions (*of*, *into*, etc.)). They cannot be changed by the user
- *predefined fixed phrases* (e.g. *there is a ...*, *it is not the case that ...*), and
- *content words* such as objects, events, qualities, properties in the outside world which are defined by the user (e.g. nouns (*John*), verbs (*enters*), adjectives (*readable*), and adverbs (*manually*)) of over 100.000 entries. Users may temporarily add their own content words.

All ACE sentences are correct English sentences but not the other way around. ACE sentences may be *simple* or *composite*. The general structure of a simple declarative ACE sentence is shown below.

```
subject + verb + complements + adjuncts
```

List 5-1: ACE sentence general structure

Every sentence has a *subject* and a *verb*. *Complements* may comprise *direct* or *indirect objects* whereas *adjuncts* (e.g. adverbs, prepositional phrases) are optional. An example of a simple ACE sentence taken from Fuchs, et al. (2009) is shown below:

```
subject + verb + complements + adjuncts

A customer inserts a card manually into a slot
```

List 5-2: A sample of a simple ACE sentence

Composite sentence are built around simple sentences by using coordination (and, or), quantification (*there is, some, every, for every*, *etc*), negation (*there is no, not, no*, *etc*), subordination (*if… then*). An example, taken from Fuchs, et al. (2009) is shown below:

[21] http://attempto.ifi.uzh.ch/site/

```
If a customer inserts a card
   that is valid
then the automatic teller accepts the card
   and displays a message
```

List 5-3: A sample of a composite ACE sentence

The Attempto Project provides a parser that evaluates automatically if a sentence is an admissible ACE sentence. This parser is called the *Atttempto Parsing Engine* (APE)[22]. APE parses ACE sentences into smaller units in the form of words and phrases (which can either be noun, verb, adjective or prepositional phrases). Rules in the composition of these phrases are also governed by ACE construction rules. Fuch, et al (2009) provides a sample of how an ACE text can be parsed into words and phrases by APE in Figure 5-12.

ACE sentences also support quantification such as: "*There is a card*" (existential quantification), or "*John enters every card*" (universal quantification). Conversely, it also supports negation such as: "*John enters no code*" (negated existential quantifier), or "*John enters not every code*" (negated universal quantifier). ACE sentences can also be interrogative in two ways: yes/no queries ("*Is the card valid?*"), or wh-queries ("*Which customer enters a card?*"). Another powerful feature of ACE is ability to resolve ambiguities in natural language when sentences have plurals or anaphoric references.

ACE sentences do not depend on any specific application domain. It does not have any knowledge or ontology of the specified domain, of software engineering methods, or of the world in general. Thus users must explicitly define domain knowledge. For example, in the sentence "*John is human*", the words are mere syntactic elements and any interpretations of these words depend entirely on the user.

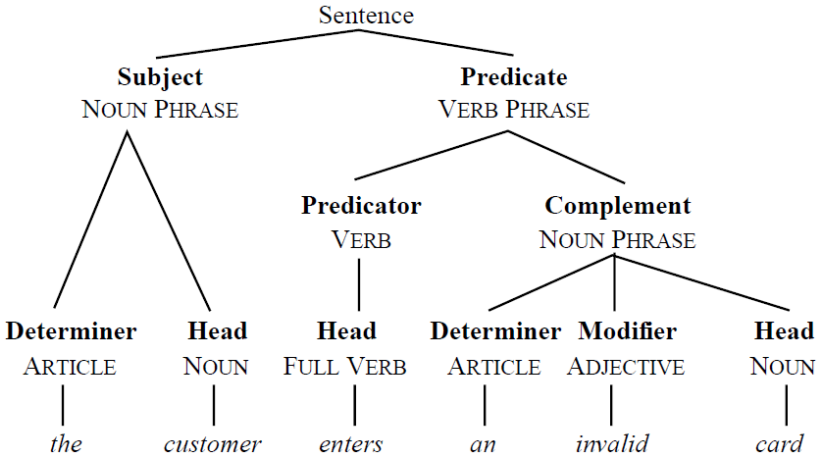


Figure 5-12: Parsing of ACE sentence in APE (Fuchs, et al., 1999)

5.2.3. RuleML

Rule Markup Language[23] (RuleML) is a markup language for publishing and sharing rule bases on the World Wide Web. It is spearheaded by the Rule Markup Initiative group whose goal is to make RuleML as the canonical XML-based markup language for rules language that permits Web-based rule storage, *interchange*, retrieval, and firing/application. RuleML has been used for deductive, reaction, and production rule specifications (Braye, et al., 2006).

RuleML is based on Datalog – a language that bridges database query languages and logic programming (e.g. SQL and Prolog) (Ceri, Gottlob and Tanca, 1989). To better understand

[22] http://attempto.ifi.uzh.ch/site/tools/
[23] http://ruleml.org/

RuleML, we first give a short overview of concepts in Datalog and then describe how they are equivalent in RuleML

Following the concepts of logic programming, a Datalog program consists of a set of *facts* and *rules*. Facts are assertions about a relevant piece of the world. An example of a fact in natural language: "*The sister of John is Mary*" or "*Mary is the sister of John*", would be written in Datalog as (c.f. List 5-4):

```
sister(John, Mary)
```

List 5-4: A sample fact in Datalog

Facts are stored in tables. In the above example, the name of the table would be `sister` while the items inside the parenthesis are columns or *terms* which may be of two types: *a logical variable* or an *individual constant*. In the case above, the terms are both individual constants. As logic facts are represented as relational tables, they can be used to reason logically about new facts based on queries.

RuleML can represent a marked up representation of facts in Datalog. In RuleML, the fact in List 5-4 would be represented as (c.f. List 5-5):

```
<Atom>
  <Rel>sister</Rel>
  <Ind>John</Ind>
  <Ind>Mary</Ind>
</Atom>
```

List 5-5: A sample fact in RuleML

The `<Rel>` tag is a *relation* tag which represents the table name of the fact, in this case the `sister`. At the same level, the `<Ind>` tag is an *individual constant* tag which represents the table's column names, which in this case are `John` and `Mary`. All of these tags are enclosed under an *atomic* formula represented by the `<Atom>` tag which in Datalog would correspond to the entire fact.

Rules are sentences which allow deduction of facts from other facts. A sample rule in natural language: "*If X is a sister of Y and if Y is a sister of Z, then X is a sibling of Z*" would be written in Datalog as (c.f. List 5-6):

```
sibling(Z, X) :- sister(Y, X), sister(Z, Y)
```

List 5-6: A sample rule in Datalog

A rule in Datalog is divided into two parts: the left-hand side (LHS) and the right-hand side (RHS) separated by the implication symbol `:-`. The LHS side is called the *head* of the rule while the RHS side is called the *body* of the rule. Also in the given example above, the terms are all *variables* to make the rule more generic.

The rule in List 5-6 can be correspondingly marked up in RuleML as shown List 5-8(next page). Here the head is enclosed by the `<Head>` tag while the body is enclosed with a `<Body>` tag. Since the RHS part of the rule contains two facts, an `<And>` tag is used to denote this. The `<Implies>` tag is equivalent to the `:-` implication symbol in Datalog. The logical variable in Datalog is represented using the `<Var>` tag.

5.2.4. Jess

Java Expert System Shell[24] or Jess (Friedman-Hill, 2008) is a *rule engine* and *scripting language* written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in

[24] http://www.jessrules.com/

Livermore, Canada, in 1995. Jess traces its roots from the *C Language Integrated Production System*[25] (CLIPS) – a programming language for the development of expert systems based on LISP, but it has from then on evolved on its own. As Jess is Java-centric, it gives systems access to Java's APIs making it easier for such systems to have added rule reasoning capabilities. Jess follows a LISP-like syntax where functions are enclosed in parenthesis. Jess has been used for deductive, reaction, and production rule specifications (Braye, et al., 2006).

```
<Implies>
  <head>
    <Atom>
      <Rel>sibling</Rel>
      <Var>Z</Var>
      <Var>X</Var>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>sister</Rel>
        <Var>Y</Var>
        <Var>X</Var>
      </Atom>
      <Atom>
        <Rel>sister</Rel>
        <Var>Z</Var>
        <Var>Y</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

List 5-7: A sample rule in RuleML

Rule-based programming in Jess is *declarative* in nature. In *procedural* programming languages like Java, the programmer tells the computer what to do, how to do it, and in what order. Declarative programming languages, however, describe what the computer should do without giving instructions on how to do it. A runtime system called a *rule engine* performs the necessary control flow logic that uses declarative information to execute instructions. SQL is another example of a declarative programming language.

As with other expert systems, Jess uses *rules* that can be repeatedly applied to a set of *facts* about the world. Jess maintains a collection of memory elements that represent information about the world called *facts*. A *working memory* contains all facts that have been loaded into RAM at runtime (thus a fact may also be viewed as a *working memory element*). In Java, a working memory is equivalent to the *object heap* where instances of Java objects temporarily reside. In essence, a fact is therefore the smallest unit of information which can be added to or removed from the working memory. For example, a person's name for a system that handles employee's information in an organization can be considered a fact. Since Jess inherits features from LISP, facts are stored as *lists*. An example of an *ordered* list containing a list of facts is:

```
(cities enschede amsterdam rotterdam)
```

List 5-8: A sample fact in Jess

There are several built-in *functions* in Jess that one can use to manipulate facts and rules. We discuss two of them: `assert` and `retract`. `Assert` adds a fact into the working memory. Asserting the code in List 5-8 would look like the code shown in List 5-9:

```
(assert (cities enschede amsterdam rotterdam))
```

List 5-9: Asserting a fact in Jess

[25] http://clipsrules.sourceforge.net/

Viewing a representation of asserted facts in working memory, we use the function `facts`. The asserted fact in List 5-9 would like:

```
f-0    (MAIN::cities enschede amsterdam rotterdam)
```

List 5-10: Viewing the working memory using `(facts)` function in Jess

Conversely, an asserted fact may be removed from working memory using the `retract` function, and the ID of the fact which in List 5-10 is `f-0`. List 5-11 shows how to retract a fact in Jess. Another way to remove a fact from working memory without using the fact ID is through *pattern binding* which we shall discuss later.

```
(retract 0)
```

List 5-11: Retracting a fact from working memory in Jess

Other built-in functions in Jess include: `clear` which removes all facts and rules in working memory, `deffacts` which defines the initial contents of working memory, `reset` which initializes the working memory, and `watch` which prints diagnostics after some event.

Facts by themselves are really not interesting Jess, they must be brought together to form rules. A *rule* is a kind of instruction or command that applies in certain situations. Rules can take action based on the facts that have been asserted into the working memory. Jess supports *forward-chaining rules* (as opposed to backward-chaining) which are somewhat like `IF…THEN` statements. However, it must not be interpreted in a procedural way where the execution is rather *deterministic*; that is, the programmer must indicate at a specific time and order when the `THEN` part should execute after the `IF` part is satisfied. In Jess, however, the THEN part is executed or *fired* whenever the IF part is satisfied, and thereafter Jess decides the order of execution based on state of the working memory.

In Jess, a rule is defined by using the `defrule` construct which divides the rule into two parts: *left-hand side* (LHS) or the `IF` part and *right-hand side* (RHS) or the `THEN` part separated by the `=>` operator. The RHS can only contain *patterns* while the LHS can only contain *actions*. The general syntax is given in List 5-12.

```
(defrule rule-name "optional comment"
    (pattern-1); left-hand side (LHS) of the rule
    (pattern-2)
    (pattern-n)
     =>
        (action-1) ; right-hand side (RHS) of the rule
        (action-2) ; consisting of elements after the "=>"
        (action-m)
)
```

List 5-12: Generic rule syntax in Jess

An example of a rule in Jess is shown in List 5-12.

```
(defrule flat-tire-rule "if tire is flat, change it"
    ?isflat <- (tire-is-flat)
     =>
       (change-tire)
       (retract ?isflat)
)
```

List 5-13: Sample rule in Jess

The LHS contains patterns which are used to match the facts that are currently asserted in the working memory, while the RHS contains actions or function calls that execute or *fire* when a matching pattern is found. Form the example in List 5-13, the pattern is `tire-is-flat`, while the actions are `change-tire` and `retract`. When the fact `tire-is-flat` appears in the working memory (after it is asserted elsewhere in the program), the *rule fires* calling the actions `change-tire` and `retract`. The variable `?isflat` is called a *pattern binding*. The `<-` operator

stores a reference to the fact `tire-is-flat` to the variable `?isflat`. When the `retract` function is called, the fact is removed from working memory without using its `fact-id`.

The rule in List 5-13 can only fire when the `run` function is called to the Jess engine. Should there be another rule whose pattern is `change-tire`, then that rule will fire. The `run` function terminates when it cannot find matching patterns in the working memory. Calling `run` again will cause Jess to do nothing. Rules are activated once only for a given working memory state. Once the rule has fired, it will not fire again for the same list of facts; that is, the `change-tire` function will not be called unless the `tire-is-flat` is asserted again elsewhere during runtime.

5.3. The Enterprise Architecture

We envision in Figure 5-13 the *generic* Enterprise Architecture to describe our solution modeled in ARMOR + ArchiMate.

We model the motivation of the integration solution through goals using ARMOR in the value layer. The integration goal which may either be a hard or soft goal can be refined further through the GORE activities described in 3.4. ARMOR provides the ability to refine goals as use cases which can also be used as one of refinement options. Modeling goals through use cases gives us the ability to determine who the stakeholders of the integration are, and the behavior expected of them. It also tells us the behaviors that a system should have in order to satisfy the goal without getting into the details as to how the goal is to be implemented by the underlying system.

Goals must be refined until they can be expressed using the requirement construct in ARMOR. Requirements in ARMOR are goals that can be assigned to a system to satisfy the goal. This can be represented in ARMOR through the realization relation between a requirement and a business service at the business layer. For example, to satisfy the `Requirement B`, it must be assigned to the `Business service A`. Thus executing `Business service A` satisfies the `Requirement B`.

If a requirement in ARMOR can be refined further so that it constrains the business process in some manner, it can thus be represented as a business rule in ARMOR. In order for the business rule to actually constrain the business process, it has to be translated into an executable form, exposed as a service and deployed to a rule engine (as discussed in Section 5.2).

At the application layer, the business services are realized by the application components of the collaborating systems. The business rules exposed as a service (named rule service) are realized by a rule engine. For example, the `Business rule A` can be satisfied if the `Rule Service A` that realizes it is invoked. The `Rule Service A`, in turn, is given its concrete realization by the `Rule engine A` at the application layer. The collaboration between the application components and the Mediator at the application layer is modeled in Arhimate through the interaction of their application interfaces.

At the business layer, several activities can each make use of the business and rule services provided by the application layer (represented through to use relation of Archimate). Together these business activities are composed through some business logic that forms the business process. The relationship between business rules and business activities can be modeled through the `<<contrain>>` relation.

Notice that the Mediator application component in the application layer also exposes a service called `Business Service M` as the Mediator is in itself a service. The Mediator service is usually used as the starting business activity in the business process that is triggered by some event. The business activity that uses the Mediator service thereafter initiates the choreography of the business services through the business activities that use them.

At the technology layer, the application components in the application layer are realized by physical devices where these components are deployed.

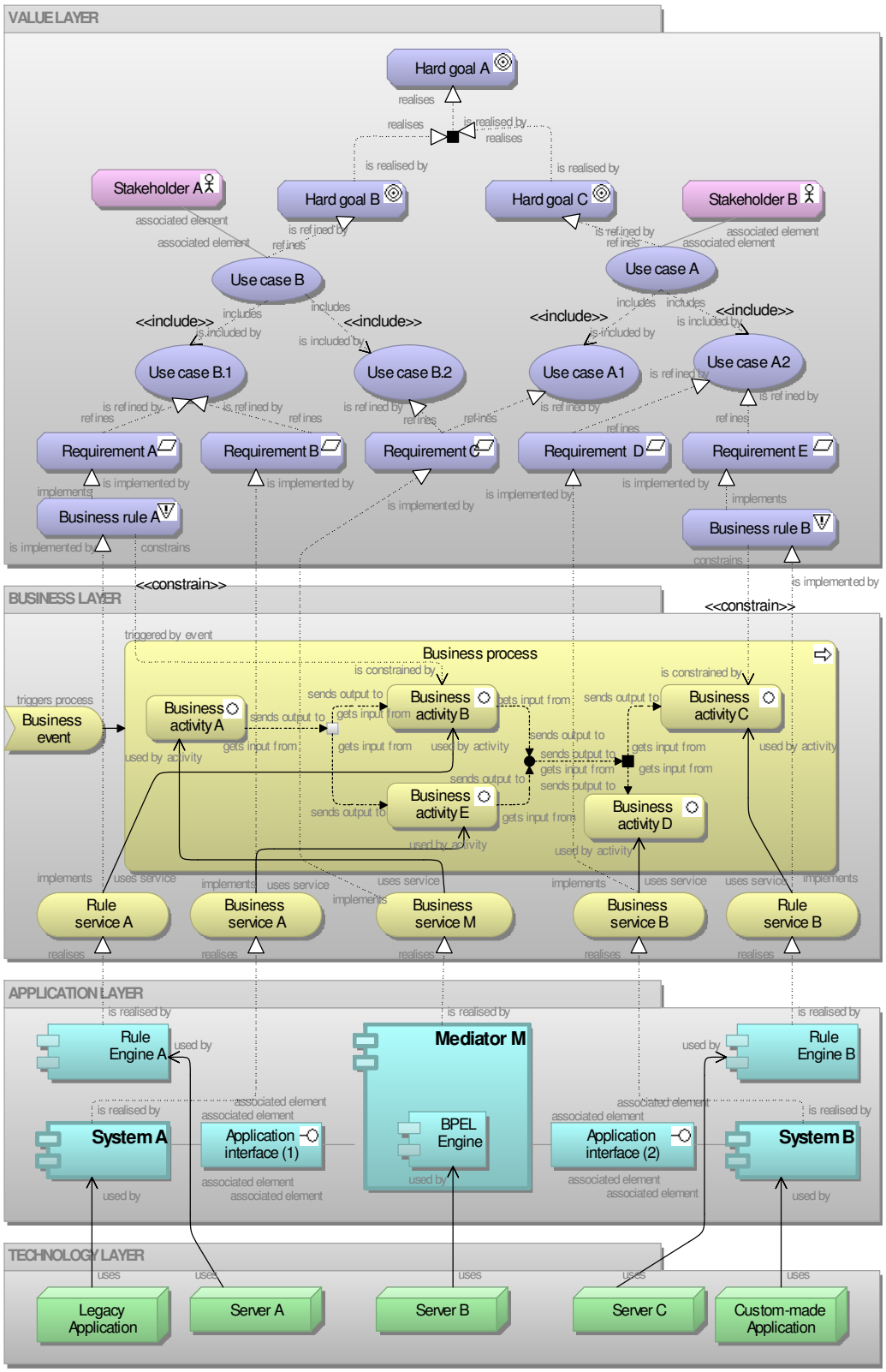


Figure 5-13: Generic service mediation architecture in ArchiMate

5.4. The methodology

Figure 5-14 shows an overview of the methodology. In describing a step in the methodology, we divide it into *What, Why*, *How* and *Who* subsections. The *What* describes the objective that the step aims to accomplish; this is meant to be interpreted *abstractly*. The *Why* describes the motivation and importance of performing the step. The *How* describes the technology and tools that we have used to *concretely* achieve the objectives of the step. We define the methodology in terms of the problem domain, rather than in terms of solution technologies. As such, although the technologies and tools we used are specific to our solution, they can be substituted by other alternatives as may be deemed ideal or appropriate by those who wish to implement the solution. For now, we focus only on describing each step of the methodology, and mention only in brief the tools necessary to accomplish the steps. In Chapter 6, we provide a detailed discussion of the functionalities of these tools. The technologies have already been described earlier in Section 5.2. Finally, we state whether a business domain expert(s), business analyst, and/or an IT expert(s) can best perform the step under the *Who* subsection.

We first give an overview of the methodology and then its details in the subsections that later follow. Figure 5-24 (page 84) shows a graphical summary of the methodology that shows the input and output deliverables for every step, including the division of the activities between the business domain expert, business analyst, and IT expert.

Methodology overview

The methodology starts by "lifting" service description described in the WSDL files into a more abstract platform independent level. This involves transforming Platform Specific Models (PSMs) to Platform Independent Models (PIMs). Doing so avoids unnecessarily complicating the design space and thus providing more opportunity for business domain experts to be involved in the design. Business domain experts don't need to understand WSDLs to design the integration solution.

The second step involves semantically enriching PIM information which cannot be automatically derived from the WSDL. This is done because WSDLs do not inherently provide interaction protocols (i.e. how the sequence of message execution is specified) so that this information is supplemented through some text documentation in natural language, stakeholder interviews, or even code inspection. This enrichment is done so that the PIM is designed completely and precisely allowing better reasoning and generation of the mediation solution later on.

As an addition to the original COSMO methodology for service mediation, we add two steps: goal modeling at the CIM layer (third step), and transformation of business rules derived from the goal models into an executable form (fourth step):

In the third step, we introduce goal modeling to engineer the requirements of the *integration solution* using ARMOR. As Figure 5-14 shows, we stress that the goal model depicts only the motivations of the *integration* at the CIM layer (represented by the single rounded square under the Mediator column). Although the collaborating enterprises may have their own private goals, we are only concerned with the goals of the integration. This stresses the importance of the needed joint collaboration of the different business domain experts from participating enterprises to identify, specify and resolve conflicts of the integration goals at the business level. From the resulting goal model, the refined sub-goals are then modeled using ARMOR's *requirement* construct and mapped onto existing services identified in Step 1 – depicting the notion that such requirements will be realized and satisfied by the existing services. Also from the goal model, business rules that constrain some aspect of the integration solution are identified. The business rules may have to be transformed into an executable form, exposed as a service and integrated into the behavior model of the Mediator (as described in the fourth step) if no existing service can realize or satisfy them.

The fourth step transforms the business rules derived from the third step into their equivalent specifications at the different layers of the MDA stack. At the CIM layer, we state the business rules in ACE where they are specified in near-natural English. The ACE rules are then transformed into an XML-based specification using RuleML for added rule interoperability. Finally, at the PSM layer, we transform RuleML into an executable form using Jess rules. The Jess rules are then exposed as a Web service through Java so that they can be added in ISDL to constrain the behavior model of the Mediator.

The fifth step involves the actual design of the Mediator at the PIM level. This usually involves splitting the integration solution in two areas which may be done in parallel: generating the *behavior* and *information* models. The information model unifies the differences in the data representations and interpretations between systems. The behavior model composes requested and provided mismatching service by relating their operations (i.e. matching the input of an operation call to the output of another and their constraints). This composition includes the Jess rules that have earlier been exposed and deployed as a Web service in the fourth step.

The sixth step involves validation of the composed service Mediator using techniques such as interoperability assessment (Quartel and Van Sinderen, 2007b) and the simulation of generated behavior and information models.

The final seventh step includes transforming the designed PIM models of Mediator to PSM as its technology implementation so that the Mediator can be made available for production use.

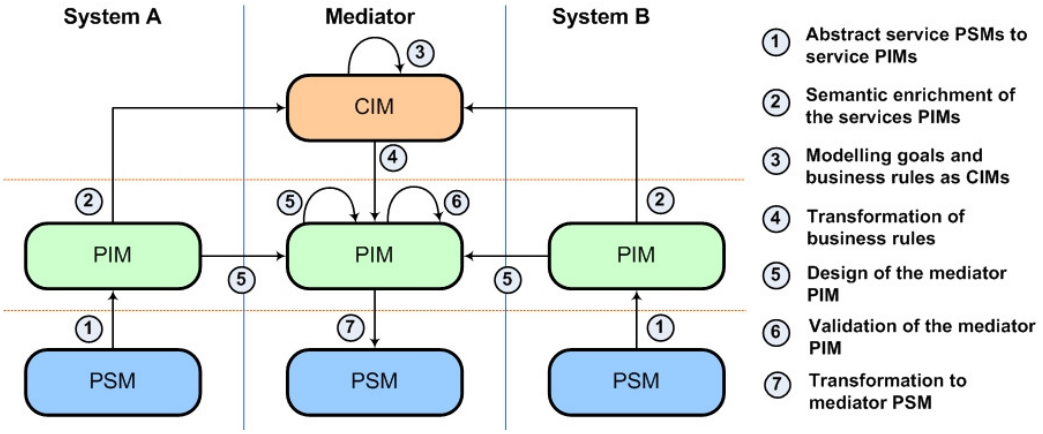


Figure 5-14: Service mediation methodology

We now describe each step in detail.

5.4.1. Step 1 – Abstracting from PSMs to PIMs

What

We begin by gathering the existing services exposed by the collaborating systems described through their WSDL documents. The objective of this step is to "lift" the *platform specific information* and *behavior models* (i.e. the messages and operations) from the WSDL documents into *platform independent information* and *behavior models*.

Why

The abstraction process is important as this will avoid direct manipulation of WSDL files in composing the service Mediator. In effect, this allows us better understanding of the services without having to understand their technical representations which may unnecessarily complicate

the design of the integration solution. This essentially allows us to manipulate services at a platform-independent, semantic level.

How

For our solution, we represent the platform independent *information* models as UML class diagrams (for visualization) and Java (for execution). The *behavior* models are represented using ISDL. Figure 5-15 describes this graphically.

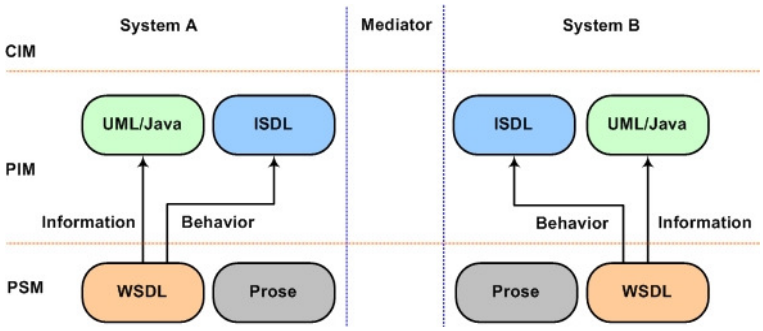


Figure 5-15: Abstracting from PSMs to PIMs

Deriving the PSM from PIM behavior models is done automatically through the WSDL import function provided by the Grizzle tool (cf. Section 6.1.2). Grizzle provides an integrated editing and simulation environment for ISDL. Given the URL of the WSDL specification, an operation, a single port type information or the entire WSDL specification can be imported to represent the perspective of either a client or server. Accordingly, the ISDL behavior model represents the Web services either as *operation calls* representing client/user perspective, or *operation executions* representing server/provider perspective.

Figure 5-16 shows an example of the abstracting WSDL operations in ISDL taken from the previous COSMO solution to the SWS Challenge (Quartel, et al., 2008a). `SystemA`'s `receiveRequest` operation is modeled to act as a client to its integration with `SystemB`, and hence is modeled as an operation call. `SystemB`'s `addLineTime` operatoin is modeled as a provider so that it is modeled as an operation execution.

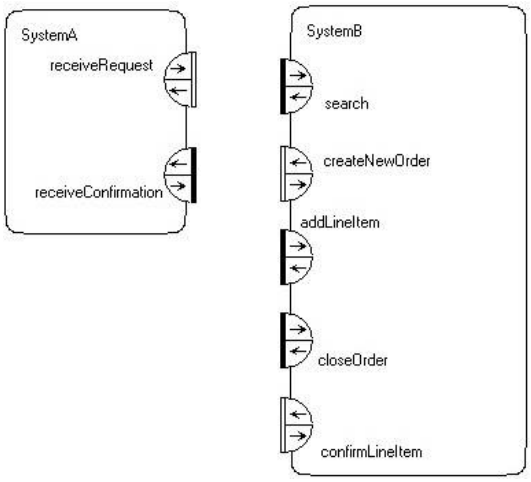


Figure 5-16: Abstracting operations in ISDL (Quartel, et al., 2008a)

Deriving the PSM from PIM information model is done by generating Java classes which represent the information types that are referred to by the operations in the behavior model. The transformation of WSDL to ISDL and Java is implemented using JAXB or JAX-WS. The open

source Omondo EclipseUML (cf. Section 6.1.5) tool can be used to visualize and manipulate the information model using UML class diagrams.

Who

We recommend that this step be done by IT experts.

5.4.2. Step 2 – Semantic enrichment of PIMs

What

The objective of this step is to add semantic information to the behavior and information models that were lifted from the WSDL specifications in the previous step.

Why

A WSDL specifies details about the operations and types of input and output messages of the Web service. But, it does not provide information about how invocations of the WSDL operations are to be executed in some order; that is, WSDLs do not provide *interaction protocol* information. Similarly, WSDL data types use XML Schema to represent the messages of the operation which only describe its *syntax*. Additional *semantics* of the message elements will have to be taken from domain specific knowledge. These pieces of information have to be derived elsewhere. The enrichment is done so that the PIM is designed completely and precisely allowing better reasoning and generation of the mediation solution later on. We represent this additional information as the boxes labeled "Prose" in Figure 5-15.

How

Adding interaction protocol information to the generated ISDL behavior models means defining how operation calls are related to the operation executions, and how values of their parameters depend with one other. This information may need to be extracted from textual descriptions of relevant documentation (such as business logic and business rules), interviews with relevant technical and business stakeholders, or perhaps code inspection of existing systems. For example in Figure 5-17, the `receiveRequest` operation must be executed first before the `receiveConfirmation` operation as depicted by the enabling condition (i.e. arrow)

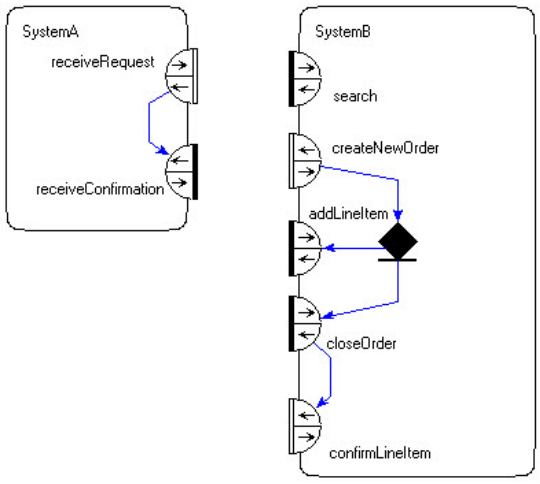


Figure 5-17: Semantic enrichment of behavior model in ISDL (Quartel, et al., 2008a)

On the other hand, the generated UML/Java files may also be enriched by similar textual descriptions, interviews or code inspection (by evaluating their class relations). The semantics of

the messages may also need to be derived from business domain experts who have domain specific knowledge of the meanings of the message elements. This collaboration is all the more important when business domain experts belong to different organizations each having their own interpretations of semantically alike but syntactically different message structures. Additionally, the meanings of the classes and their properties can be defined by mapping them onto some domain-specific ontology. This step is done manually.

Who

We recommend that this step be done mainly by IT Experts in consultation with business domain experts and business analysts.

5.4.3. Step 3 – Model goals and business rules at CIM

What

The objective of this step is to capture the motivation or rationale of the *integration* where business requirements are specified as goals or business rules without first describing how they are implemented by the underlying systems.

Why

As we have described in Section 3.3, specifying business requirements through goal modeling concepts provides a natural way for identifying, structuring, clarifying, refining, and reasoning about the requirements (including related risks) among various stakeholders at a level of abstraction that is sufficient for business-level analysis. The role of business domain experts is crucial in modeling the goals of the integration particularly when enterprise from different domains collaborate (e.g. when a hospital aims to connect their systems with an insurance firm to handle payment of health insurance claims). Goal models also provide a mechanism to directly connect requirements into their technical implementations allowing better business-IT alignment and requirements traceability.

How

Figure 5-18 graphically shows how this step will be achieved (modeled using a business process modeling tool called BiZZDesigner[26]). Briefly, this step begins with the business domain experts from collaborating enterprises working together and driving the requirements engineering process by identifying, refining, and resolving the goals of the *integration* solution. With the assistance of business analysts, they refine the goals using ARMOR until such a point where some of the sub-goals are refined enough to be assigned or matched to the existing service identified in Step 1. Some goals may be refined into business rules and stated in plain English. If no existing service can realize the satisfaction of the derived business rules, then they will have to be translated into an executable form and incorporated in the behavior model of the Mediator.

Next, the IT expert takes the business rules written in plain English and translates them into ACE sentences manually. This step is necessary since ARMOR does not currently support automatic transformation of business rules into ACE (or any controlled language for that matter)

Finally, the translated business rules in ACE will have to be validated again by business domain experts. This is done simply by visually inspecting if whether or not the semantic equivalence is preserved between the business rules stated in plain English and those that were stated in ACE. This step iterates until the correct semantic equivalence is achieved. Figure 5-18 graphically this step is to be performed.

[26] http://www.bizzdesign.nl/joomla/products/bizzdesigner.html

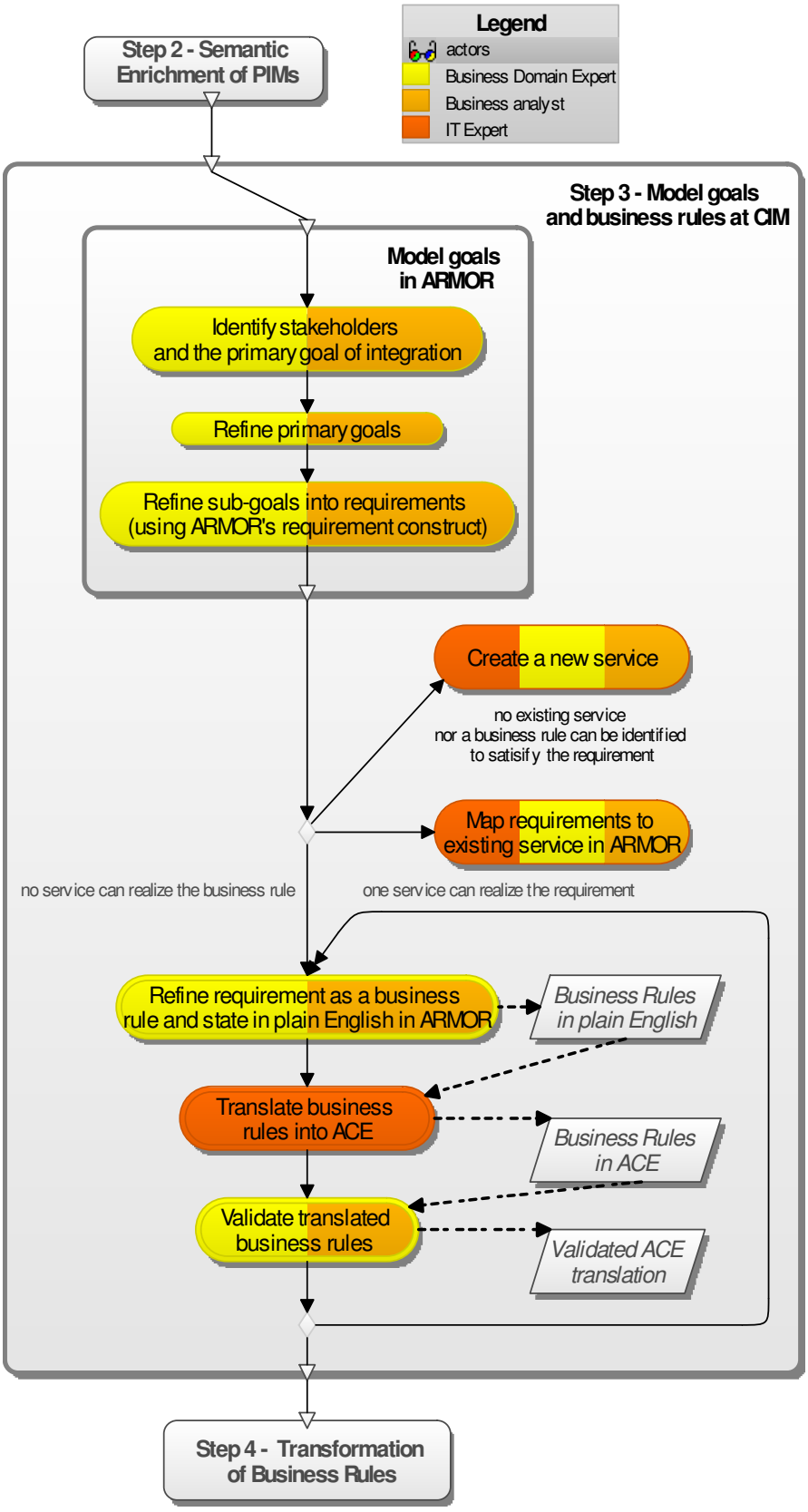


Figure 5-18: Modeling goals and business rules at CIM (modeled in BiZZDesigner)

Substep 1: Model goals in ARMOR

To model goals of the integration in ARMOR, we adopt the suggested methodology by Mantovaneli Pessoa, Van Sinderen and Quartel (2009)[27]. We divide this substep into three steps: (i) identification of stakeholders and their primary goals, (ii) refinement of primary goals, and (iii) manual refinement of sub-goals into requirements.

Identification of stakeholders and the primary goal of integration

This step essentially identifies all stakeholders of the integration and their primary goal of the integration. A goal identification activity can be performed through interviews or searching for intentional keywords if the initial business requirements are stated in some preliminary documents, and other techniques described in Section 3.4.1.

Refinement of primary goals

Next, we refine the primary goal of the integration by performing goal decomposition and refinement activities as described in Sections 3.4.2 and 3.4.3 by repetitively asking *how* questions to the primary goal and *why* questions to the derived sub-goal. In ARMOR, use case models may be also be used to refine the primary goal where each use case can be assigned to an actor of the integration solution. Use case relations like include or extend relations may be used to refine the parent use case into finer child use cases.

Refinement of sub-goals into requirements

Finally, we manually refine the goals obtained from steps 1 and 2 into more concrete and specific sub-goals. Goal refinement, in principle stops, if a sub-goal can be depicted as a requirement; that is, a single agent (or in our case, an exiting service identified in Step 1) can be used to satisfy the achievement of that requirement.

Substep 2: Map requirements to an existing service in ARMOR

From the refined goal model in ARMOR, we manually match which existing service can be assigned to a requirement so that such service when executed can satisfy the achievement of the requirement. We depict this correspondingly in ArchiMate using the *realization relation* construct.

Substep 3: Refine requirement as business rule

If no service can be identified to satisfy the requirement modeled in ARMOR, but there are some business requirements that can be derived to constrain the requirement, we model these business rules using the business rule construct in ARMOR and state them in plain English.

Substep 4: Create new service

We assume in this thesis that all services needed in the mediation already exist. However, if no system can satisfy the requirements (as modeled in ARMOR) or that no business rule can be derived from the requirement, then a new service may have to be created. This is, however, beyond the scope of this thesis.

[27] As an alternative to this goal elicitation methodology, van Laamswerde (2009, p. 309 - 326) provides an approach to building goal models using the KAOS methodology which starts from (i) *eliciting preliminary goals* where initial goals are identified using goal identification techniques described in Section 3.4.1, (iii) identifying goals along refinement branches using goal abstraction and decomposition techniques described in Sections 3.4.2, 3.4.3 and 3.4.4. The KAOS approach also provides some *tips to avoid common pitfalls* during the goal modeling process and an extensive collection of *goal refinement patterns*.

Substep 5: Translate business rules into ACE

Since ARMOR does not currently support the transformation of its business rules meta-model into a controlled language such as ACE, we need the IT expert to manually translate such business rules into their equivalent ACE sentences. Business rules in ACE are usually specified in `IF … THEN` statements (cf. Section 3.5). To facilitate the manual translation of plain English text to ACE, we recommend the use of the Attempto Parsing Engine (APE) Web client whose functionalities and use are described in Section 6.2.1. APE parses sentences to determine if they are valid ACE using ACE's construction and interpretation rules (cf. Section 5.2.2)

Substep 6: Validate ACE translation

Once the business rules have been translated as valid ACE sentences using the APE Web client, the ACE sentences will have to be shown again to business domain experts for validation. Here, the role of the business domain expert is to *manually* compare the translated ACE sentence from the business rule modeled in English to determine if the translations bear the same meaning. This should be a doable process since ACE sentences are written in a form that is intuitive and naturally understandable. Once the ACE sentences have been validated, we move to the next step which is to transform these ACE rules into an executable form as described in Step 4.

Who

As Figure 5-18 suggests, we recommend that this step be done in close collaboration between the business domain expert, business analyst, and the IT expert.

5.4.4. Step 4 – Transformation of business rules

What

The objective of this step is to translate and deploy business rules derived from the goal model into an executable form.

Why

As we have stated in Section 3.5, separating the business rules from business processes they constrain allows us to treat them transparently; that is, they are no longer hidden in some application code or scattered elsewhere in the organization. This separation of concerns essentially keeps our solution designed for change – should there be changes to the business rules, the impact to the underlying systems should not be too significant.

Furthermore, the separation allows business domain experts (who created them in the first place) to manage and update them efficiently, independent of the rest of the applications that implement them. Business domain experts should be able to identify how the business rules are implemented by the underlying system so that should there be any changes, they can assess the impact at the business level.

Since the business rules are initially defined in near-natural language at CIM level, several transformations have to be done to translate them into their PIM and finally PSM equivalents. In the end, the executable business rule will be added to constrain the behavior model of the Mediator.

How

We accomplish this step through the following consecutive sub-steps:

Substep 1: Transformation of ACE to RuleML.

Transforming ACE to RuleML requires the use of the *ACE2RRML* Web client described in Section 6.2.2. The validated ACE sentence is simply entered into the text area of the user interface to derive the RuleML equivalent of the ACE sentence.

Substep 2: Transformation of RuleML to Jess Rule.

We have created a prototype to transform RuleML into Jess rule expressions using XSL Transformation (XSLT). To execute the transformation, open source or commercial tools may be used. Section 6.2.3 provides more details of the transformation. No changes to the generated RuleML are necessary.

Substep 3: Deployment of Jess rules as web services.

Our next step is to wrap Jess rules as a Web service so that they can be added to and accessed by the behavior model of the Mediator in ISDL at the PIM level and BPEL at the PSM level. Section 6.2.5 details the architecture of the deployment.

Using our RuleML to Jess XSLT transformation, the generated Jess rules are not readily executable. Platform-specific Jess codes are must be added to allow Jess to be invoked via Java as a Web service. To facilitate the coding of Jess rules, the JessDE editor can be used whose functionalities are described in Section 6.2.4 .

Who

We recommend that this step be done mainly by IT experts.

5.4.5. Step 5 – Design of the Mediator PIM

What

The objective of this step is to design Mediator PIM in ISDL based on the operations modeled in ISDL, the messages of the operations modeled in UML classes (discussed in Step 2), and the business rules that that have been deployed as Web services in Jess (discussed in Step 4).

Why

We perform this step to serve the purpose of the mediator which is to match process and data mismatches between collaborating systems.

How

We accomplish this step through the following consecutive sub-steps:

Substep 1: Importing Jess rules as a Web service in ISDL

At this point, the Jess rules, exposed as Web services, are still described through their WSDL specifications. We thus need to perform again Step 1 as described in Section 5.4.1 where we abstract the platform-independent information models in UML /Java and the behavior models in ISDL (as operation calls or operation executions). We do this using the import functionality of the Grizzle tool. After this step, we are now ready to begin designing the information and behavior models of the Mediator.

Substep 2: Matching provided and requested services

This step matches in ISDL the provided service offered by one collaborating system to the request service of another system. We do this by taking the "complement" by changing the

operation call in one system into an operation execution in another, and vice versa, while keeping the same parameters. Analogously, the services that are requested by the Mediator can be obtained by taking the complement of the services. This results into a skeleton of the Mediator. The relation between the provided and requested services can be found by matching the input that is required by each operation to the output that is produced by other operations. Matching operation parameters is currently done manually. Furthermore, specific processing logic may have to be designed manually. Figure 5-19 shows an example of the Mediator skeleton. In the Mediator's side, the `receiveRequest` operation is modeled as an operation execution which is taken as the complement of the `receiveRequest` operation call modeled on the `SystemA`'s side.

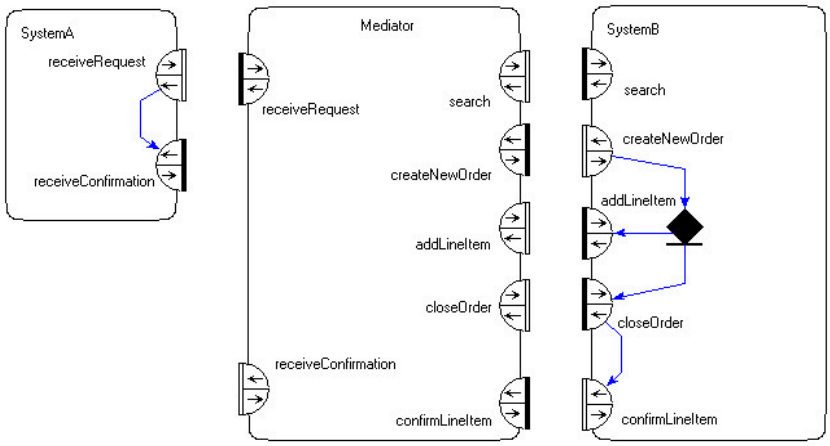


Figure 5-19: Matching provided and requested services (Quartel, et al., 2008a)

Substep 3: Composing services by relating their operations

After complementing the operations on the mediator side, service composition can begin. This can be done by matching the input information required by an operation to the output operation that is provided by the other. For example, if the `search` operation shown in Figure 5-19 requires some information from the message of the `receiveRequest` operation, a causal relationship between the two operations can thus be established. Figure 5-20 shows the enabling condition (i.e. depicted as an arrow) that relates the `search` and `receiveRequest` operations together.

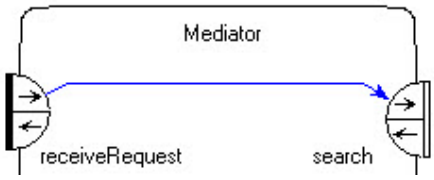


Figure 5-20: Composing services by relating their operations

However, matching input and output operations may not be sufficient to compose services especially in situation where processing logic may have to be designed explicitly and manually. In such case, the requirements documents, interviews, flow charts may be used as sources of information.

Substep 4: Transforming data among the operation parameters

From the matched requested and provided services in Substep 2, the message parameters of their operations will also be matched. This requires a definition of their data transformations describing how each input parameter is generated from the values of the output parameters.

The data transformation will have to be handled by the Mediator. A Domain-Specific Language called *MDSL* has been developed for this purpose where a set of mappings functions perform data transformation between two or more objects.

Data transformation is currently done manually. To facilitate the coding of data transformation using MDSL, we have created a prototype tool editor called Tizzle based on the Eclipse Plugin Text Editor Framework. This tool is described further in Section 6.1.4.

Consider the example in Figure 5-21, using the custom mapping function `messageA2messageB`, `Systems A` tries to send an XML-based message `MessageA` that must conform to the semantics of `System B`'s `MessageB` structure. The appropriate semantic equivalence between message elements is indicated by the arrows. In the MDSL code of the data transformation, the `source` clause uses a common variable (e.g `varK`) to match the equivalent message under the `target` clause. In the example, the source `ElementX` of `MessageA` is matched to the target `ElementC` of `MessageB` via the variable `varK`. The meanings between these messages are derived from domain-specific knowledge, and are usually defined by the business domain experts.

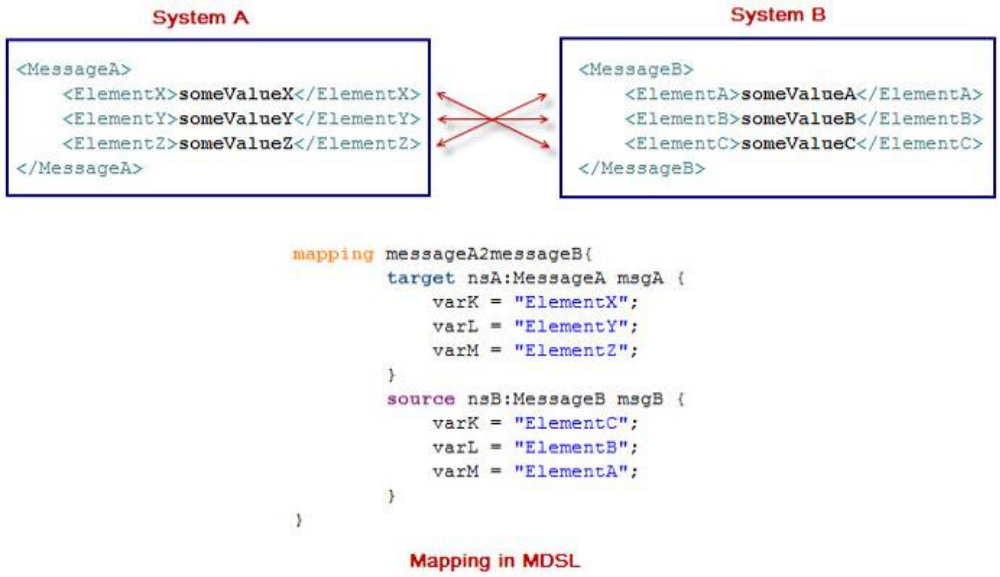


Figure 5-21: Sample data transformation mapping in MDSL

Who

Designing the behavior model of the Mediator is largely technical in nature since IT experts need to have knowledge of service-oriented design principles such as service coupling, cohesion, parsimony, etc. (Erl, 2005). Furthermore, they have to knowledgeable of service-oriented modeling and refinement patterns in ISDL as proposed by the COSMO approach (Quartel, et al., 2007a). We thus recommend that this step be done by IT Experts.

5.4.6. Step 6 – Validation of Mediator PIM

What

Once the process and data mismatches have been completely designed, the next step is to validate the ISDL Mediator model before it is deployed for production use.

Why

Validation allows us to verify whether the Mediator will behave as expected, see how interactions of operations evolve at different conditions, evaluate the ability to of the Mediator to meet certain requirements (e.g. performance, resource utilization, throughput times, etc.), and to verify if the Mediator does indeed meet the specified business requirements (Narayanan and McIlraith, 2002). Validation also increases the confidence stakeholders have on the Mediator model and the results it produces (Robinson, 1997).

How

The COSMO methodology for service mediation currently supports two ways of validation of the Mediator PIM in ISDL: *behavior simulation of services in ISDL*, and *interoperability assessment of services*. This research focuses more on behavior simulation. We discuss the second option briefly.

Behavior simulation of services

The simulation the Mediator in ISDL is supported by the Sizzle tool (cf. Section 6.1.3). Sizzle provides the ability to analyze orderings of the occurrences of operation calls and executions modeled in ISDL, including evaluation of the resulting established information. It allows real-time invocation of operations by using automatically-generated stub codes as hooks to execute real Web service invocations. The operation results are also incorporated during the simulation for analysis. It can also be used by other external web clients (e.g. test beds) to invoke the Mediator PIM for simulation purposes. Sizzle, however, does not support other important execution properties such as performance, monitoring, etc (necessitating the transformation of the Mediator PIM to its BPEL process specification as described in the next step).

Interoperability assessment of services

Quartel, et al. (2007b) proposes a method for interoperability assessment of service composition which is composed of two steps: (i) checking the results established by individual interaction, and (ii) checking whether the entire service composition can establish a result.

Checking whether an individual interaction produces a result can be done by evaluating the common effect established by the operation call and operation execution that satisfies both of their constraints. For example, if a provider requires that a user should be able to pay via credit card, and that the provider can deliver a pizza within the vicinity of the user, only then that the online-pizza-order interaction result can be established.

Checking the result of the whole service composition can be done by viewing the interactions of the collaborating systems in an integrated perspective. The operations are treated as joint actions and transformed into a Colored Petri Net. The Petri Net is then used to construct occurrence graph to conduct reachability analysis. The analysis checks whether operations are reachable in a certain order.

Who

Even if the Mediator PIM is syntactically or structurally correct, it does necessarily mean that the Mediator PIM has captured the specified business requirements. The knowledge of business domain experts is thus of paramount importance to determine if the business logic, business rules, and other requirements have been met (Sadiq, Orlowska, Sadiq and Foulger, 2004). We thus recommend that this step be done by IT experts in coordination with business domain experts and business analysts.

5.4.7. Step 7 –Transformation to Mediator PSM

What

In this final step, the PIM of the Mediator specified in ISDL is transformed automatically to PSM specified in BPEL. The generated BPEL specification is then deployed and executed on a standard BPEL engine. We briefly describe the steps involved. For a more detailed treatment of the transformation, we refer you to Dirgahayu, et al., (2007) and Quartel, Dirgahayu and Van Sinderen (2009b).

Why

This step is necessary so as to make the Mediator available for production use.

How

A language called *Common Behavioral Patterns Language* or CBPL has been developed to enable reusable transformations between business process models and their implementations. To achieve such reusable transformations, CBPL is used to decouple the definition and implementation of pattern recognition and pattern realization from the target implementation language (Dirgahayu, et al., 2007). CBPL has been applied in transforming ISDL to BPEL.

The transformation begins first by manually annotating the ISDL operations with BPEL-specific information that is used as input to the transformation. The stereotype maps abstract ISDL behavior constructs onto concrete BPEL constructs and other information needed at PSM level (e.g. BPEL partner links). These annotations are represented as stereotype information in the ISDL operation. Figure 5-22 shows an example of BPEL-specific stereotype annotations of the AddLineItem behavior.

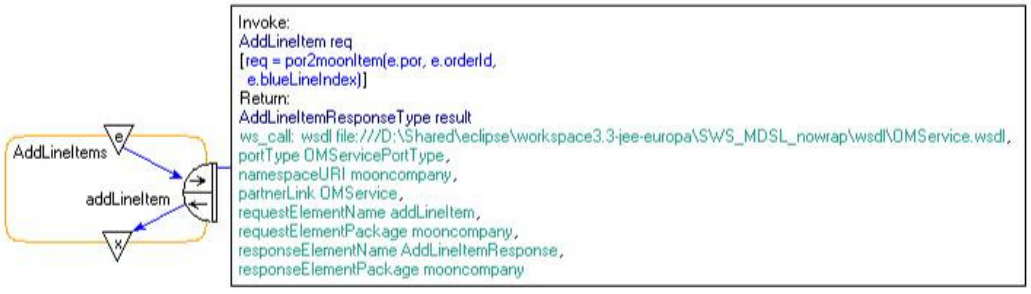


Figure 5-22: Adding BPEL constructs as stereotypes to ISDL

The second step requires the recognition of common behavior patterns and their translation to the basic patterns in CBPL such as sequence, concurrence, selection, and iteration. The final step realizes these basic patterns into their equivalent constructs in BPEL such as `bpel:sequence`, `bpel: while`, `bpel: flow`, and `bpel:if`. Figure 5-23 shows an example of an ISDL model transformed into CBPL and BPEL.

Who

We recommend that this step be performed by IT experts.

Figure 5-23: Sample ISDL to CBPL to BPEL transformation (Dirgahayu, et al., 2007)

5.5. Chapter summary

We involve business domain experts in the design of mediation solution by providing them concepts and tools to state their requirements using ARMOR. We have chosen ARMOR largely because of its suitability for the design of enterprise architectures, ability to document, communicate and reasons about requirements, and provisions for an easy-to-use set of language constructs. Its integration with the Archimate modeling framework for Enterprise Architecture also allows us to perform better business-IT alignment and requirements traceability analyses. Unlike any other goal modeling language described in Section 3.6, ARMOR + Archimate provides use a high-level integrated view of the integration architecture from the requirements down to the implementation

We extend the COSMO framework for service mediation by introducing goal-driven approaches. Using ARMOR, we specify how the composition will be achieved by first modeling the goals at the CIM level through goal analysis/identification, decomposition/refinement, or abstraction techniques. We then use ArchiMate to model how these goals are implemented and realized by the underlying Enterprise Architecture.

Once we have the requirements of the service mediation solution expressed in terms of goals driven by business requirements, we use the framework proposed by Iacob, et al. (2009) for goal-driven design of service-oriented systems using model-driven techniques. The framework seeks to incorporate a business view in SOA development where high-level goals are refined and operationalized as business rules. These rules are then transformed into a language where it can be executed and eventually incorporated into the design and composition of services.

We treat business rules as design artifacts and separate from the business process they constrain. This keeps the business rules transparent (e.g. they are no longer hidden in some application code). Exposing the executable rules as services allows them to be distributed, reusable and accessible.

We have kept our methodology problem-oriented allowing implementers of the solution to choose their own technologies. For example in the Design Space, ISDL may be substituted by other PIM business process technologies/standards such as Business Process Modeling Notation (BPMN)[28], UN/CEFACT's Modeling Methodology (UMM)[29], etc. Other goal modeling approaches may also be used as described in Section 3.6 although they may be insufficient if

[28] http://www.bpmn.org/
[29] http://umm-dev.org/

there is a need to see how the goal models are realized by the underlying architecture. In the Goals and Business Rules space, other technologies can also be used. At the CIM level, business rules may be specified in Semantics of Business Vocabulary and Business Rules or SBVR (Object Management Group, 2008a). At the PIM level, other XML-based rule specifications such as OMG's Rule Interchange Format (RIF)[30] may be used. At the PSM level, other proprietary rule-based systems can also be used such as IBM ILog or Oracle Business Rules.

[30] http://www.w3.org/2005/rules/wiki/RIF_Working_Group

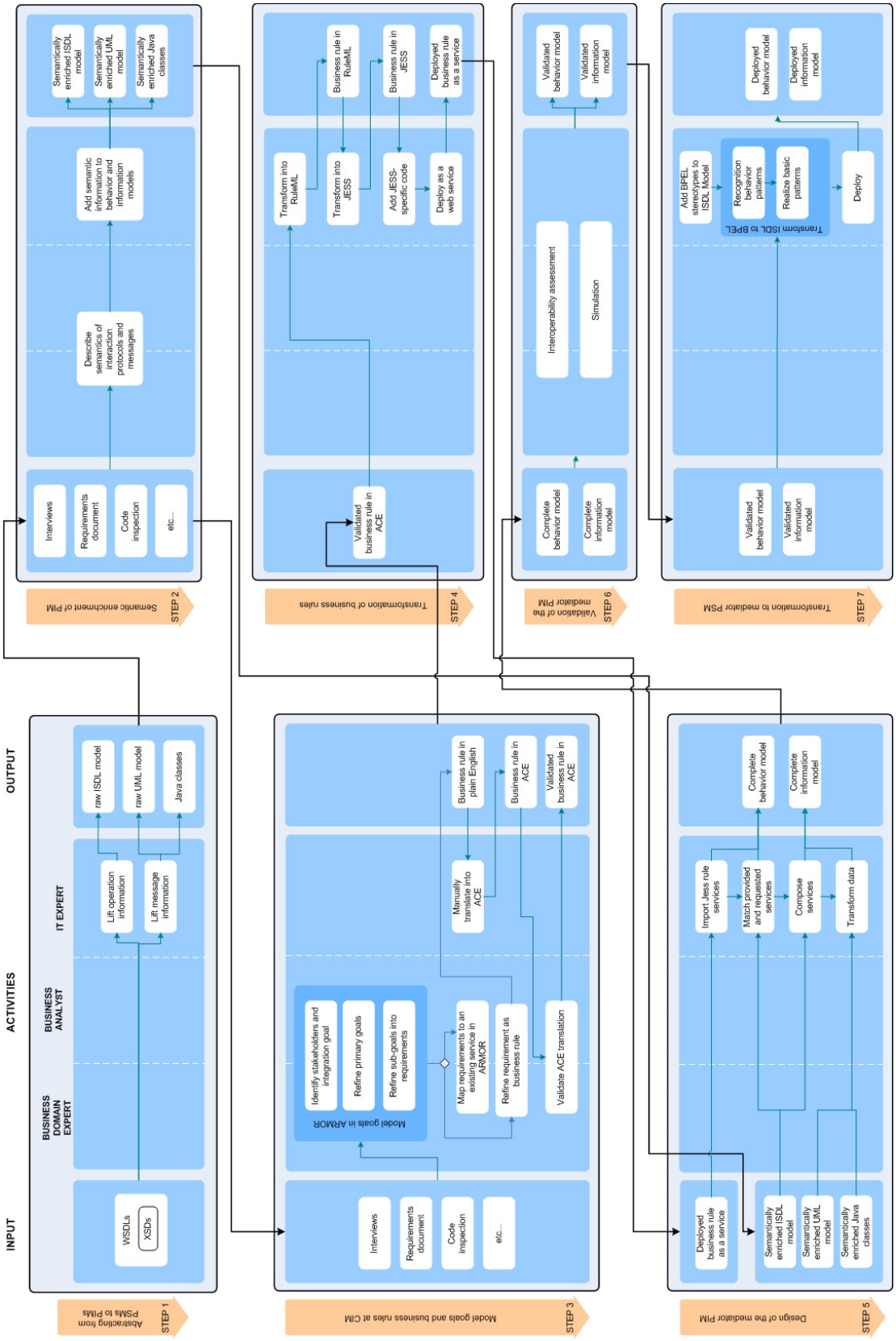


Figure 5-24: Methodology summary

6

Tool support

This chapter provides a description of the tools used in the development of the mediation solution, including tools for the transformations between the rule technologies. We divide this chapter in two usage areas: Design Space and Goal and Business Rules Space. Tools under the Design Space where used in the modeling of the behavior and information models of the Mediator. Tools in the Goals and Business Rules Space were used in the modeling, design, transformation, and execution the goals into business rules.

The structure is as follows: Section 6.1 describes the tools in the Design space which includes BiZZDesign ArchiMate Professional (Section 6.1.1), Grizzle (Section 6.1.2), Sizzle (Section 6.1.3), Tizzle (Section 6.1.4), and Omondo EclipseUML (Section 6.1.5). Section 6.2 describes the tools used in the Goals and Business Rules Space which includes the APE Web service client (Section 6.2.1), ACE to RuleML transformation tool (Section 6.2.2), RuleML to Jess transformation (Section 6.2.3), JessDE (Section 6.2.4), and our Jess rule as a Web service deployment architecture (Section 6.2.5).

6.1. Design Space

6.1.1. BiZZDesign Architect

BiZZDesign Architect[31] is a commercial tooling environment for the ArchiMate Enterprise Architecture modeling language. It is developed by BiZZDesign[32] – a spin-off company from Novay as part of the Testbed research program. BiZZDesign Architect runs on Windows 2000 to Windows Vista. The most important features[33] of BiZZDesign Architect are the following:

- *Architectural domains and relations*. Provides support for the architectural modeling of products and services, business functions, business processes, organization, applications, infrastructure, and data at several levels of abstraction. Business goals, architecture

[31] http://www.bizzdesign.nl/joomla/products/architect.html
[32] http://www.bizzdesign.nl/joomla/
[33] http://www.bizzdesign.nl/joomla/component/option,com_docman/Itemid,0/task,doc_download/gid,1/

requirements, and critical success factors can also be modeled and related to architectural models and views, effectively maintaining relationships between architectural domains.

- *Input and visualization*. Facilitates easy drawing of architecture models and views and provides a way to import tables from Microsoft Office applications (e.g. a list of applications) including functionalities for coloring and labeling of models and automatic layout functionality.
- *Viewpoints and views*. Allows user-customized definition of viewpoints for effective and efficient communication with different stakeholders.
- *Impact analyses*. Provides better impact analysis when architectural models change (may be caused by, for example, application replacement, product change, architecture principle change, or infrastructure component renewal) including the ability to identify which part of the architecture the change occurred.
- *Dependencies and landscape maps*. Provides a functionality to see direct and indirect dependencies between architectural models (e.g. 'which departments use which applications to support which business processes')
- *Publication*. Provides a functionality for architectural models to be directly published (in part or in full) on an Intranet or in Microsoft Word. Selective model publication allows relevant stakeholders to have a more precise view of the architecture.
- *Repository*. Provides support for storage of architectural models using Oracle or SQL Server, or using a shared file, allowing better version management and multi-user support.

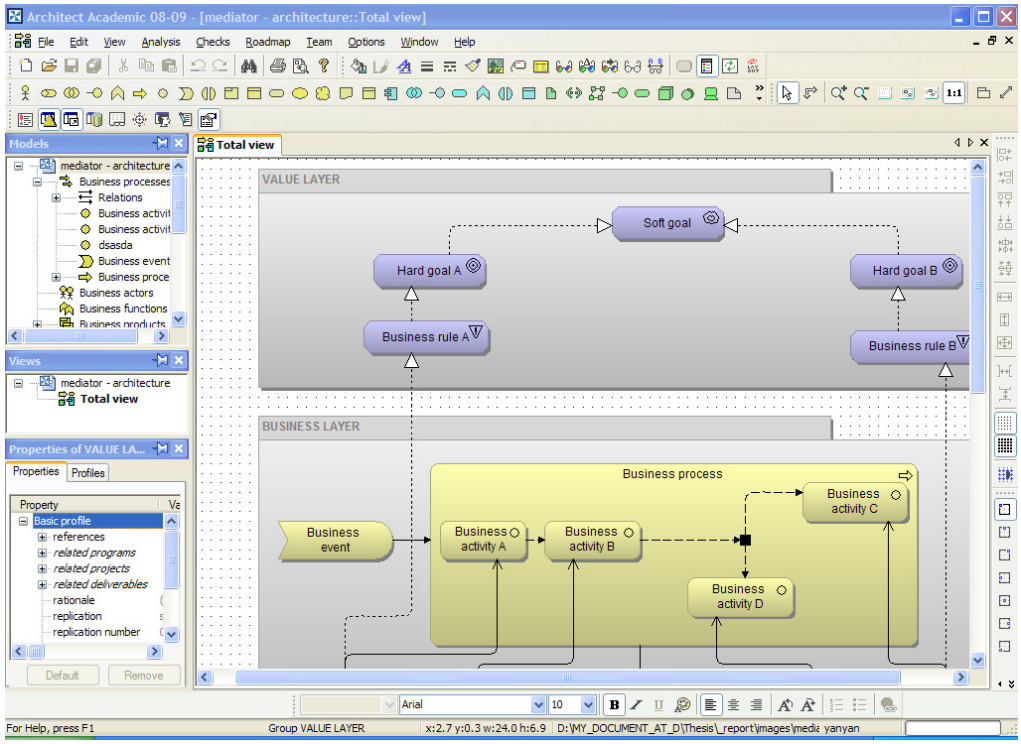


Figure 6-1: BiZZDesign's ArchiMate Professional for Enterprise Architecture modeling

6.1.2. Grizzle

Grizzle[34] (screenshot shown in Figure 6-2) is an editor for the graphical notation of ISDL developed by Novay. It allows the creation and manipulation of the graphical notations of an ISDL model. Grizzle is available in Windows, Linux and SunOS distributions. It is freely available for download (except its source code). This thesis uses version 0.73.07. Figure 6-2 shows a screenshot of the Grizzle editor.

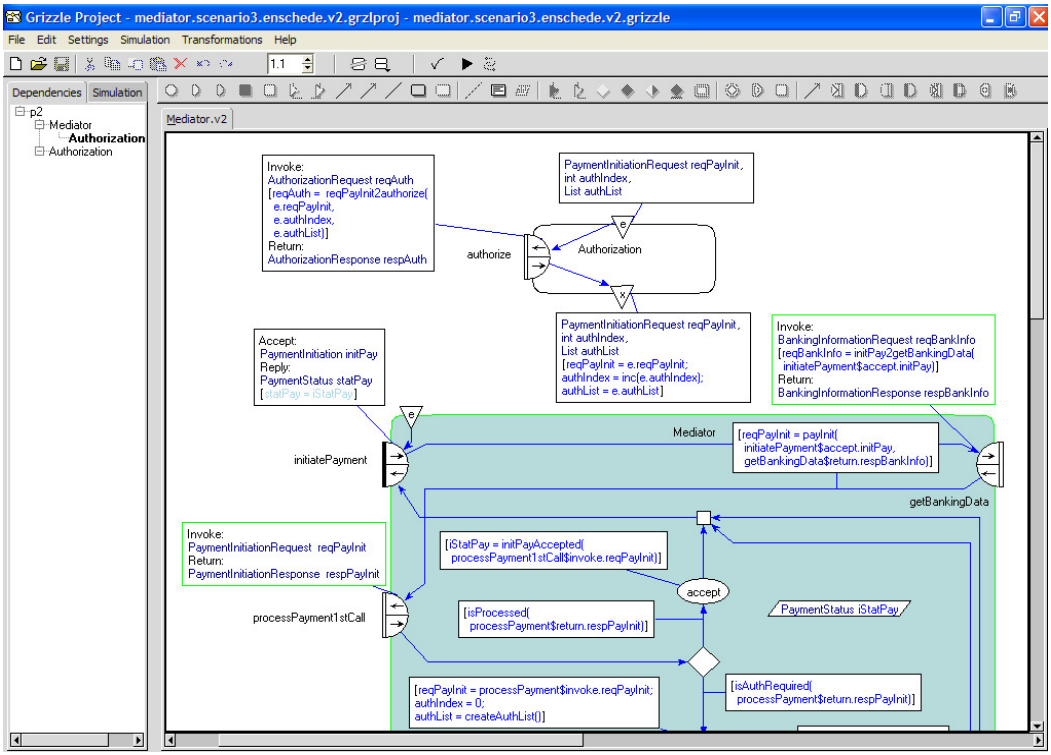


Figure 6-2: Grizzle for behavior modeling in ISDL

The following description of the basic functionalities of Grizzle is taken from Quartel, Dirgahayu and Van Sinderen (2009b).

- *Syntax checking*. Allows user-requested validation of the graphical syntax, with the ability to identify the source of error textually or by highlighting the corresponding part in the graphical representation (recommended before performing more operations on the model).
- *Model organization*. Provides the ability for (sub-)behavior models to be organized and separated into different sheets (implemented as a tab).
- *Ecore XMI export*. Allows the export of an ISDL model into an instance of the Ecore Metamodeling Language (EMF).
- *Petri Net export*. Provides a mapping from ISDL to Petri Nets in a format readable by CPNTools[35].
- *Language profiles*. Provides a functionality to create ISDL dialects (i.e. specialized or extra language elements added to the basic ISDL language) as language profiles.
- *Stereotyping*. Allows graphical language elements to be annotated with stereotypes to add specific modeling information.

[34] http://isdl.ctit.utwente.nl/tools/index.htm
[35] http://wiki.daimi.au.uk/cpntools//cpntools.wiki

- *Splitting and joining behaviors*. Allows support for transformation of behavior models by joining two interacting behavior models into a single integrated behavior model, or splitting a single integrated behavior model into two interacting behavior models.
- *WSDL import*. Allows import of a WSDL specification through its URL for Web service behavior modeling. A single operation, single port type or the complete WSDL definition may be imported.
- *Simulation front-end*. Provides a front-end for simulating ISDL behavior models.

6.1.3. Sizzle

Sizzle[36] is the simulator for behavior models in ISDL as it allows analysis of the possible behavior executions. It is freely available for download; the source code, however, is not publicly available. It is packaged together with Grizzle. Figure 6-3 shows a screenshot of the Sizzle editor.

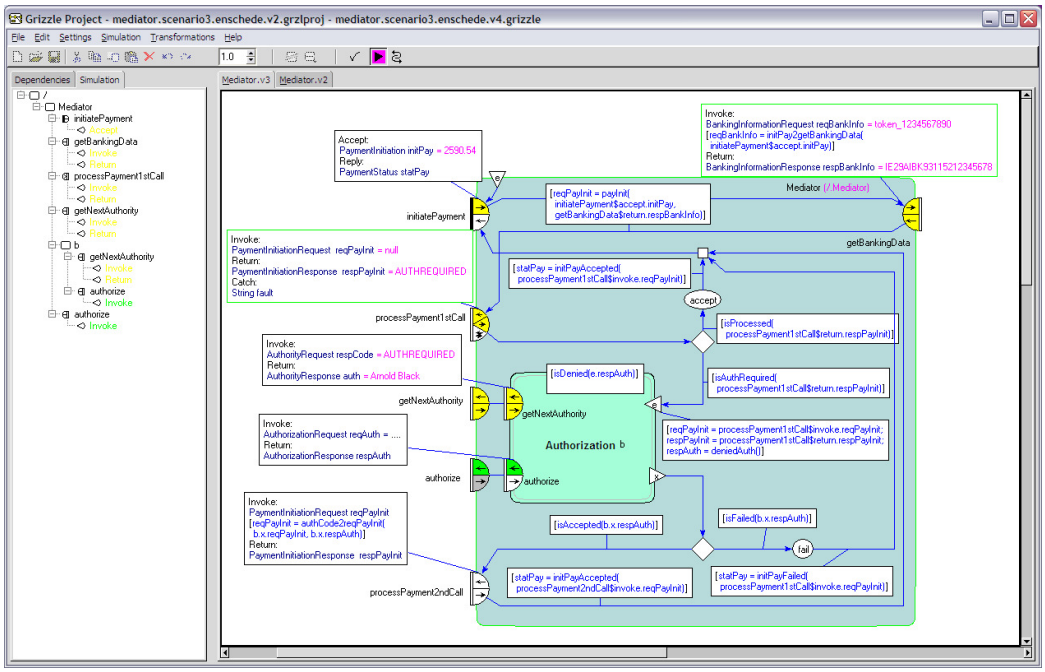


Figure 6-3: Sizzle for behavior simulation in ISDL

The main characteristics of the simulator are listed below.

- *Causality semantics*. Provides causality-based semantics of IDSL where the simulation status maintains causal dependencies between activities; that is, an activity can only refer to the results of other activities that have directly or indirectly caused its execution.
- *Interaction semantics*. Provides support to describe interaction semantics (a kind of negotiation where interaction contributions represent negotiation constraints).
- *Transformation to the basic ISDL profile*. Provides support for transformation of other profiles to the basic ISDL profile for simulation. The transformation enforces consistency and compliance of the new profile with the COSMO framework.
- *"Live" simulation of web-services*. Provides support to perform Web service invocations and incorporate results returned by the Web service in the simulation environment. A stub code (automatically generated based on the defined stereotype information which includes the

[36] http://isdl.ctit.utwente.nl/tools/grizzle/index.php

Web service's endpoint address and port type name) is linked to an interaction contribution representing a Web service invocation.

Behavior model simulation can take the following steps:

1. *The preparation step*. Based on the current simulation status, this step determines which activities of a behavior are enabled.
2. *The user interaction step*. This step allows users to select one of the enabled activities for execution. Enabled activities are colored green, and executed activities are colored yellow.
3. *The execution step*. This step executes the selected activity, and updates the simulation status accordingly, thereafter returning to the Step 1 after completing this step.

6.1.4. Tizzle

Mapping data translations using the Domain Specific Language (DSL) mapping language to solve data mismatches has previously been a complex and tedious manual process. A large part of the work was fixing syntactically incorrect mapping definitions. This problem has largely been attributed to the lack of a dedicated tool to support the process. Previously, a plain text editor was the only tool for this purpose. To mitigate this problem, we have built a dedicated Eclipse Plugin *editor* to assist developers during the mapping exercise. We call this editor *Tizzle*.

The most important features include:

- *JAXB file generation within the Eclipse environment*. Previously, generating the required JAXB files of the different XSD Schemata was largely done via the command line. We have improved this situation by putting the generation inside the Eclipse environment. This reduces development time as users just select some menus to generate the files, and don't have to type in error-prone command line parameters. Furthermore, the generated JAXB files inside Eclipse will also be used for dynamic content-assist feature described next. We borrowed the source codes from the *XJC Plugin for Eclipse 3.1.0*[37] of the *JAXB-Workshop Project*[38] owned and maintained by *Kirill Grouchnikov*[39], and customized it for our purpose.
- *Dynamic content assist*. Using the generated XML schemata inside Eclipse, Tizzle dynamically queries all possible XSD element paths with respect to the current XSD element selected by the user, and displays these in a form of a content-assist popup menu. The user only needs to select his desired element by clicking on it. Previously, the user needs to painstakingly look into the XSD files to identify the correct element, and manually type the entire XSD element's path in the editor which proved to be tedious, complex, and extremely error prone.

Other features include:

- *Code folding*. The lines of code oftentimes grow to a large number especially for large XSD message structures. The code folding feature provides user to "fold" code blocks, reducing scrolling length of the editor.
- *Syntax highlighting*. Keywords of the DSL mapping language are colored similar to standard text editors of the Eclipse Text Editor Framework (e.g. the standard Java Editor that comes with Eclipse). Aside from reducing misspelled keywords, it also allows user to easily identify code blocks.
- *Outline view*. The `mapping`, `source`, and `target` keyword constructs of the DSL mapping language are given their equivalent tree representations under the Outline View of the editor.

[37] https://jaxb-workshop.dev.java.net/plugins/eclipse/xjc-plugin.html
[38] https://jaxb-workshop.dev.java.net/
[39] http://www.java.net/blogs/kirillcool/

This is a handy feature that allows users to go to a specific code block right away without scrolling through the entire editor, providing faster code navigation.

- *Automatic code formatting.* Uses smart code indention as new line code is typed in.

Tizzle was written using Eclipse 3.4.0 Plugin Development Environment, J2SE 1.5, JAXB 2.1, and JFace Text Editor Framework (comes with the Eclipse standard installation). Tizzle is freely available including the source code. A screenshot of Tizzle is shown in Figure 6-4.

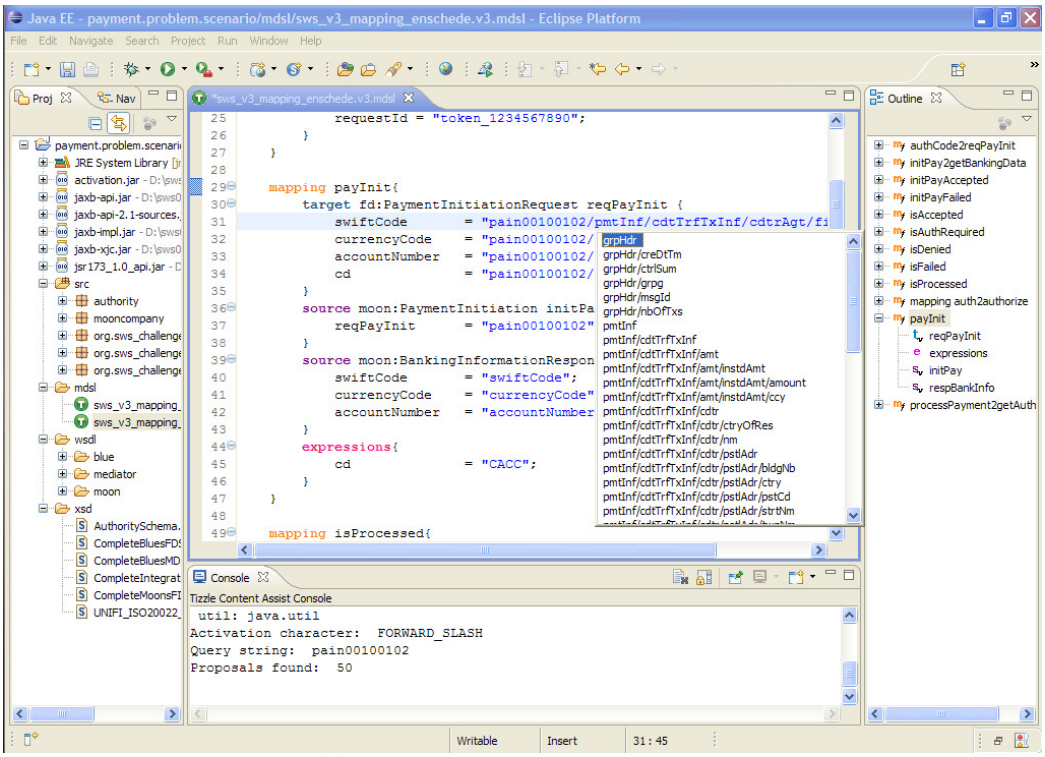


Figure 6-4: Tizzle for mapping data translations in MDSL

6.1.5. Omondo EclipseUML

From the generated JAXB files, one can use Omondo EclipseUML[40] – a commercial UML modeling environment based on Eclipse – to reverse engineering Java files into their UML representations. UML diagrams provide an easier way to understand relations between classes including their associations, dependencies, and generalizations between different packages or projects without having to read Java Code.

This thesis uses the UML diagrams of the generated JAXB files to manually relate the semantics between message elements of Web services operations. Figure 6-5 shows a screenshot of Omondo EclipseUML reverse engineering some classes. We use the EclipseUML 2008 version which needs Eclipse version 3.4, the Eclipse Graphical Editing Framework (GEF), the Eclipse Modeling Framework Project (EMF), and at least Java JDK 1.5 or higher (all comes with the installation files).

[40] http://www.eclipsedownload.com/index.html

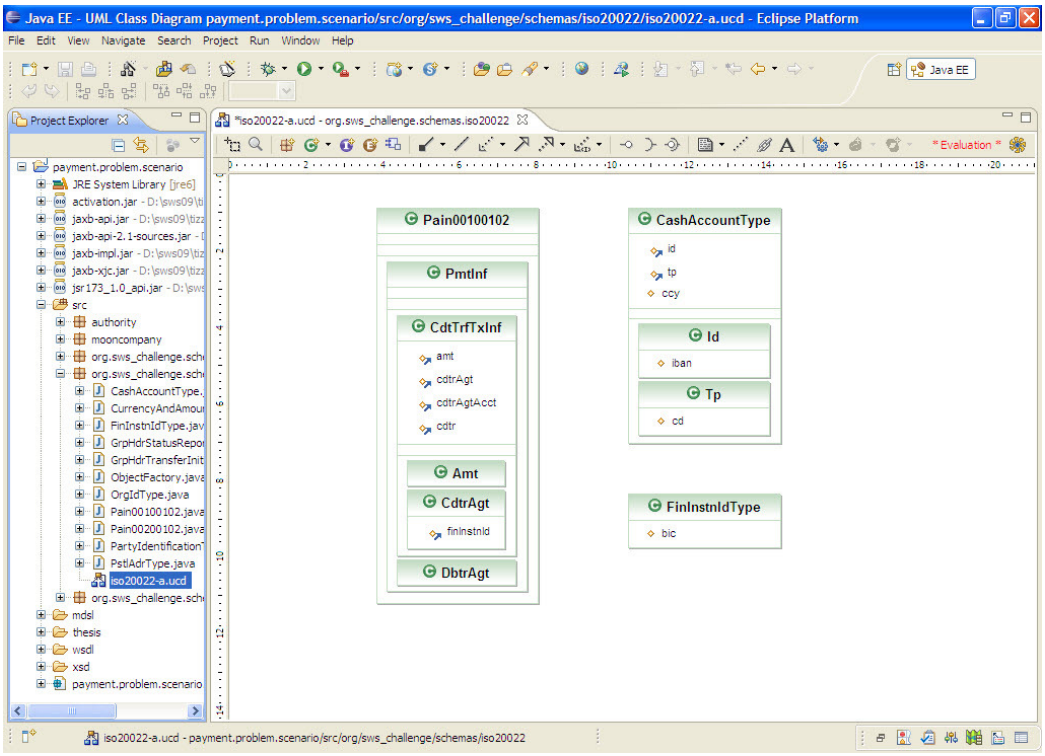


Figure 6-5: Omondo Eclipse UML

6.2. Goals and Business Rules Space

6.2.1. APE Web client

The Attempto Project provides an AJAX-based Web client[41,42] for its Attempto Parsing Engine (APE). The Web client is a front end to the APE Web service[43] (version 6.5) which translates an ACE sentence into a Discourse Representation Structure (DRS) and other logical forms.

Figure 6-6 shows a screenshot of the APE Web client user interface in action. The APE web client works in Firefox (1.0 and later), Safari (1.3 v312 and later), Opera (8.5 and later) and Internet Explorer (6.0), and tested largely Mac OS X with Firefox 3, Safari 3.1.2 and Opera 9.5. The Attempto project does not provide the source codes of the parsing engine and the Web client.

A brief description of the functionalities of the Web client is as follows:

- The ACE text area must contain either an ACE text or its web address (URL).
- The arrow buttons browse the history of previously entered ACE texts (also contains examples).
- `Analyze` sends the ACE sentence entered in the text area to APE via the APE Web service. Results are presented under the control box which later appears below the text area. Adding

[41] http://attempto.ifi.uzh.ch/site/docs/ape_webclient_help.html
[42] http://attempto.ifi.uzh.ch/ape/
[43] http://attempto.ifi.uzh.ch/site/docs/ape_webservice.html

new ACE text for analysis preserves the previous result which allows comparison of different results (via the arrow buttons).

- The `Show` checkboxes controls the display of the components of the analysis' results (e.g. paraphrase, DRS, Syntax, etc).
- The `Guess unknown words checkbox` allows APE to guess the word-class of unknown words.
- The `Do not use Clex` checkbox causes APE to ignore its built-in large lexicon of English content words which makes APE to rely on the user lexicon and guessing.
- The `Lexicon URL` option when selected causes APE to use APE only with its built-in lexicon.
- The `Reload the lexicon` causes APE to reload the lexicon from user-provided web address.

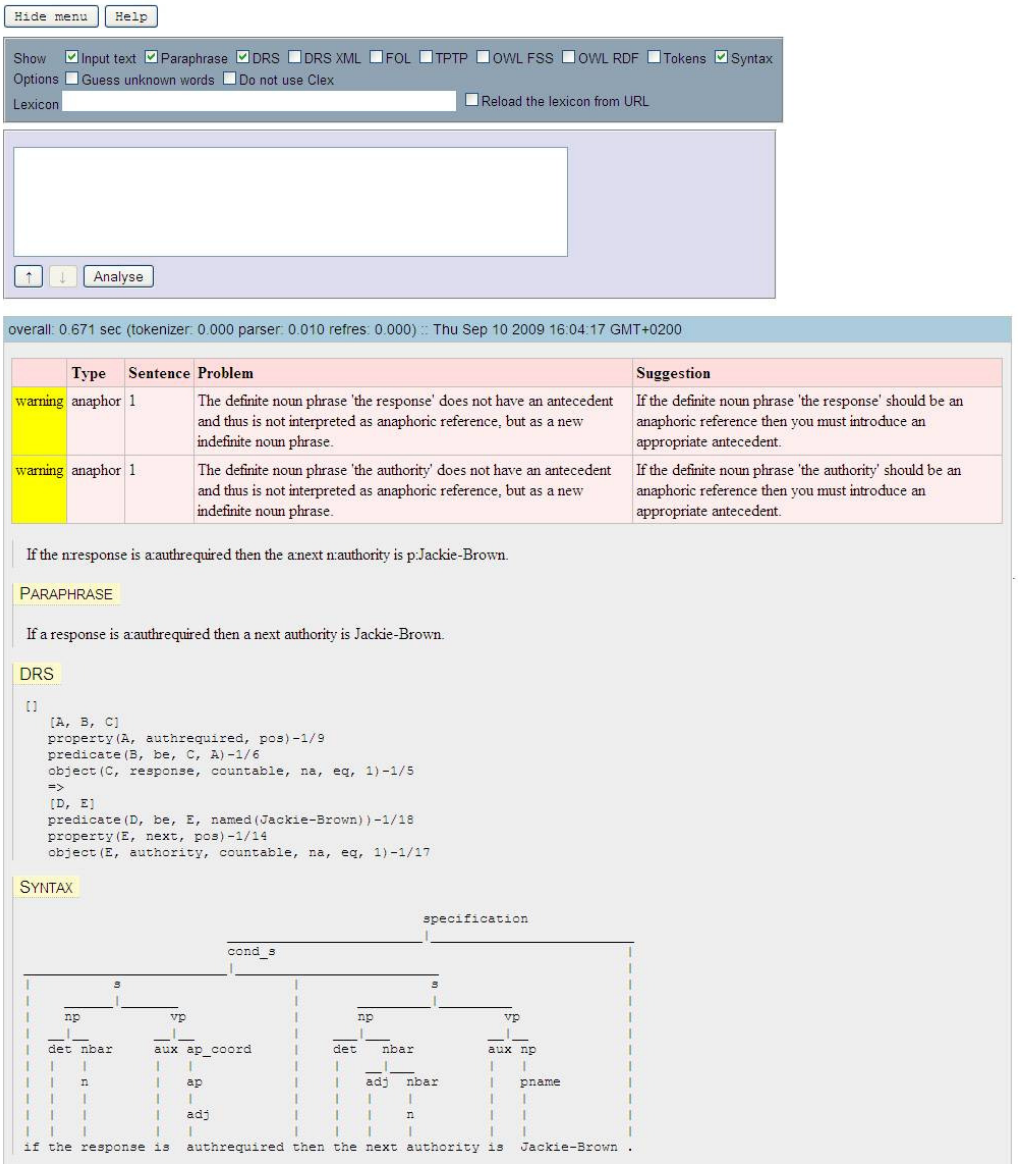


Figure 6-6: Attempto Parsing Engine (APE) web interface

Three basic steps to use the APE Web client are recommended as follows:

- *Step 1. Enter and parse an ACE text.* Using the construction and interpretation rules described in Section 5.2.2, enter an ACE text in the text area, and click the `Analyze` button.
- *Step 2. Analyze the results generated by the Web client.* If the entered sentence is an ACE text, APE shows a linguistic analysis of the input text: its tokenization, Discourse Representation Structure and syntax trees (pretty-printed in simple ASCII-graphics), and other options selected from the `Show textboxes`
- *Step 3. Recover from errors.* In case APE has problems processing the text, it shows an error or a warning message. Error messages mean that parsing has failed and that the returned DRS is empty. Warning messages do not report fatal errors but only point to potential problems in the input. Errors are presented in red color, warnings are yellow.

  One source of errors is unknown words which APE assumes it to be a typographical error. One way to teach APE about the content words not from its internal lexicon is by prefixing unknown words with the *word-class marker*, e.g.: `p:Johnny, v:inuntiate, a: humongous n:rook, a:abruptly`; where, `v` means verbs, `n` means nouns, `p` means proper names, and `a` means adjectives and adverbs. The next time these words are used again, they no longer need to be prefixed as the APE remembers them. Multiword terms must be hyphenated and prefixed by the word-class marker, e.g.: `John is a n:nonpartisan-activist`. Alternatively, users can put new words into their own lexicon, and specify the URL of the lexicon on the `Lexicon URL` field.

6.2.2. ACE2RRML

Bahr (2008) provides a prototype for translating ACE sentences into (Reaction) RuleML – *ACE2RRML*[44]. His work is similar to the work of Hirtle (2006) called *TRANSLATOR*[45]; however, the latter seems to be buggy as it does not have the same DRS output as APE. Both authors' works are available for download, including their source codes. ACE2RRML uses Apache Axis2 as Web service middleware, and can be deployed in Apache Tomcat 5.5, using J2SE 1.5 or higher. Figure 6-7 shows the user interface of ACE2RRML.

Before attempting to translate an ACE text to RuleML, we first recommend that the ACE text be parsed using the APE Web client described in 6.2.1 as it provides a much better debugging environment for ACE coding. After the text is found to be a valid ACE text by APE, the translation from ACE to RuleML should follow. ACE2RRML accomplishes the translation from ACE to RuleML in the following manner:

- *Communication to the APE Web Service.* Similar to the APE Web client described in Section 6.2.1, ACE2RRML takes the ACE text input and forwards it to APE via the APE Web service. ACE2RRML thereafter extracts the DRS component from the XML response of the APE Web service. The DRS is then fed to an XML parser
- *Translation of the DRS output into RuleML.* ACE2RRML translates DRS into either *literal* or *domain specific* RuleML representation. Literal translation involves direct translation of DRS output into RuleML syntax (similar to the work of Hirtle (2006)). *Domain specific* translation involves restricting the input language to obtain a translation that does not use the DRS vocabulary but rather that of the desired domain. The full details of the translation can be found at the project's website documentation[46]. If domain specific translation does not work the literal one is generated. This thesis uses the literal translation provided by ACE2RRML.

[44] http://www.pa-ba.info/?q=pub/ace2rrml
[45] http://ruleml.org/translator/#Doc
[46] http://www.pa-ba.info/sites/default/files/documentation.pdf.gz

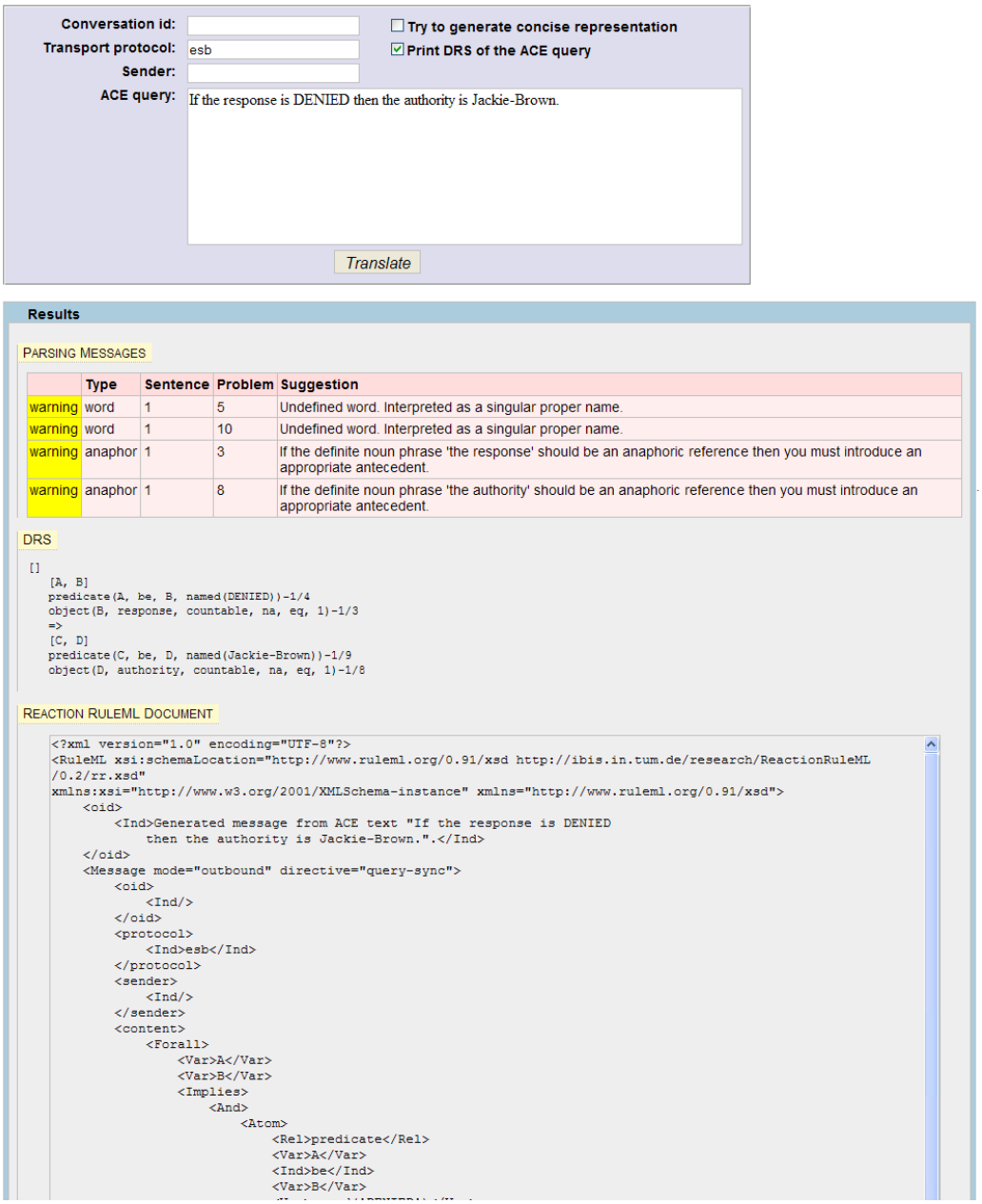


Figure 6-7: ACE to Reaction RuleML translator by Bahr (2008)

- *Produce output*. The translated RuleML representation is then embedded in an XML structure specific to ACE2RRML where it embeds other information as conversation ID, protocol, sender, etc. The output is presented to the client back in the user interface.

Now, a brief description of its user interface:

- `Oid`. The conversation id of the Reaction RuleML message.
- `Protocol`. The protocol used for the Reaction RuleML message.
- `Sender`. The sender of the Reaction RuleML message that is to be produced.
- `ACE`. The source ACE text to be translated.

- `tryConcise`. Indicates whether DRS to RuleML translation should be done either using the more sophisticated, domain-specific concise translation, or the direct verbose literal translation.
- `addDRS`. Indicates whether the output should also print the DRS returned by APE.

6.2.3. RuleML to Jess transformation

We have created a prototype for transforming RuleML to Jess rules based on the `ruleml2jess` transformation work of *Said Tabet*[47] of the RuleML Initiative. The work of Tabet was not used directly because it was based on an earlier version of RuleML (version 0.8). The version supported by ACERRML is based on RuleML version 0.9, requiring us to write our own transformation prototype. Figure 6-8 shows a comparison of the differences in versions 0.8 and 0.9 of RuleML.

```
<imp>
    <_head>
        <conclusions>
            <assert>
                <fact>
                    <atom>
                        <_opr>
                            <rel>...</rel>
                        </_opr>
                        <var>...</var>
                        <ind>...</ind>
                    </atom>
                </fact>
            </assert>
        </conclusions>
    </_head>
    <_body>
        <and>
            <fact>
                <atom>
                    <_opr>
                        <rel>...</rel>
                    </_opr>
                    <var>...</var>
                    <ind>..</ind>
                </atom>
            </fact>
        </and>
    </_body>
</imp>
```

version 0.8

```
<Forall>
    <Implies>
        <And>
            <Atom>
                <Rel>...</Rel>
                <Var>...</Var>
                <Ind>...</Ind>
            </Atom>
            <Atom>
                <Rel>...</Rel>
                <Var>...</Var>
                <Ind>...</Ind>
            </Atom>
        </And>
        <Exists>
            <And>
                <Atom>
                    <Rel>...</Rel>
                    <Var>...</Var>
                    <Ind>...</Ind>
                </Atom>
                <Atom>
                    <Rel>...</Rel>
                    <Var>...</Var>
                    <Ind>...</Ind>
                </Atom>
            </And>
        </Exists>
    </Implies>
</Forall>
```

version 0.9

Figure 6-8: RuleML differences in version 0.8 and 0.9

We use the XSL Transformations (XSLT)[48] to transform RuleML into Jess rules as plain text. We use the open source SAXON[49] XSLT and XQuery Processor (version 9.2) to execute the XSLT stylesheet because of its tokenizing capabilities. For the complete XSLT stylesheet code, please see Appendix A.5. For more details about the transformation, please see Appendix C.

[47] http://home.comcast.net/~stabet/page3.html
[48] http://www.w3.org/TR/xslt
[49] http://saxon.sourceforge.net/

RuleML does not have a dedicated editor (although there are some academic open source prototypes for Reaction RuleML like the one of Adrian Paschke called *RBSLA Editor*[50]). However, since RuleML is XML-based, any open source or commercial XML editors can be used for this purpose. This thesis uses the commercial product Oxygen XML Editor provides an XSL/XSLT debugger and execution environment for transforming XML to text using XSLT[51] (screenshot shown in Figure 6-9). The RuleML Initiative provides a set of XSD and DTD-Schemas[52] that can be used to validate RuleML instances.

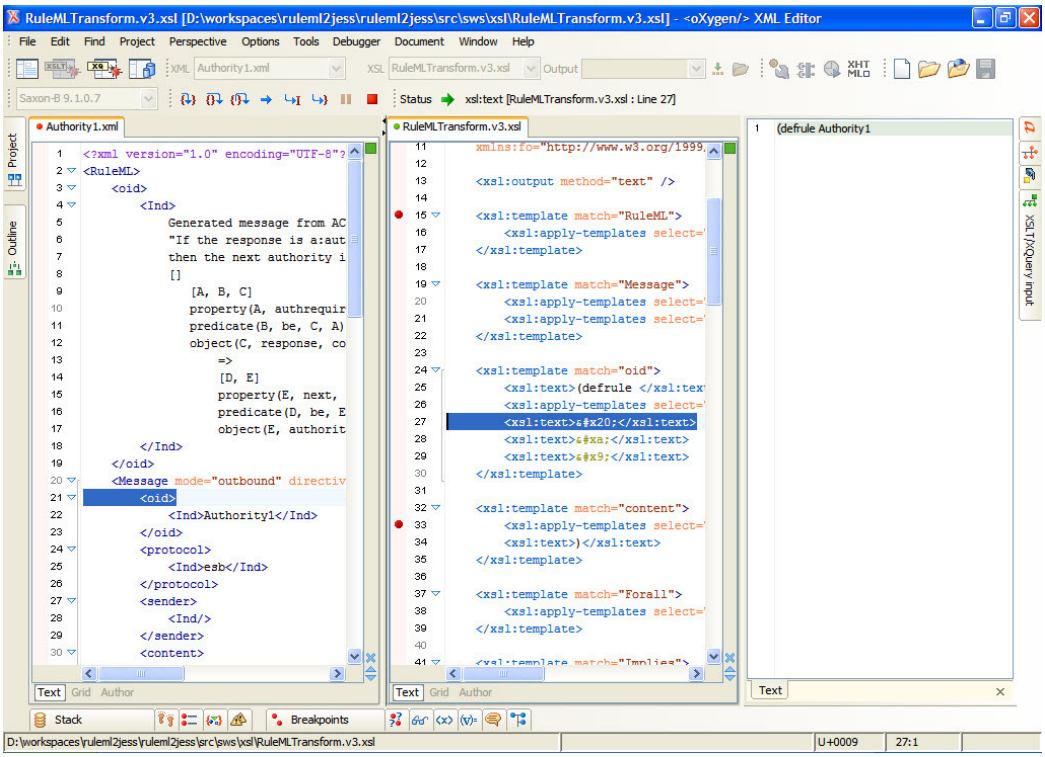


Figure 6-9: Oxygen XML Editor

6.2.4. JessDE

Jess provides a powerful, commercial-grade, Eclipse Plugin Editor for scripting and executing Jess rules called *JessDE*[53] (*Jess Development Environment*). Aside from the editor, JessDE also includes a powerful debugger, and a Rete network viewer.

This thesis uses version 7.1 of the Jess engine. Note, however, that this version is not compatible with earlier versions of the Jess engine. The Jess engine and JessDE are commercial products of Sandia National Laboratories[54] in Livermore, CA, USA. An academic license is, however, freely available. JessDE is compatible with the full SDK version of Eclipse version 3.1 or higher, and runs on J2SE 1.5 or higher. Figure 6-10 shows a screenshot of the JessDE language editor.

[50] http://sourceforge.net/projects/rbsla/
[51] http://www.oxygenxml.com/xslt_debugger.html
[52] http://ruleml.org/#DTDs-Schemas
[53] http://www.jessrules.com/jess/docs/71/eclipse.html
[54] http://www.sandia.gov/

JessDE language editor

JessDE stores Jess rules as files with a ".clp" extension. The editor has the following features common to all Eclipse-based editors.

- Customizable syntax coloring you can customize.
- Content assistant supplies `deftemplate`, slot and function names.
- "Quick fix" assistant for automatic code repair.
- Real-time error checking with markers and error highlighting while typing code
- Automatic code indent and formatting while typing code
- Outline view navigation for faster navigation of code.
- *Parenthesis matching and auto-insertion.* JessDE inserts matching character when '(' or '"' characters are typed in. Placing the cursor on a character shows where the other matching character is.
- Online help for Jess functions and constructs via "hovers" providing access to the Jess manual for every every function and construct type.
- *Help hovers for* `deftemplates` *and* `deffunctions`. JessDE shows a "tooltip" containing information and a template or a function.
- Run and Debug commands for Jess programs.
- Native XML support for reading, writing, and transforming its own XML-based rule language called "JessML" into other XML rule languages as well as into the Jess rule language.
- Addition of Java regular expressions in writing Jess rules.
- Simplified query API for searching Jess' working memory using a familiar JDBC-like interface.

Dependencies among files

JessDE allows *.clp files to be modularized. A *.clp file can be made dependent with some other *.clp using the `require*` function.

The Rete Network view

JessDE allows users to see a graphical representation of the Rete network derived from any rule using `Rete Network View`. The view displays the Rete network for the rule that the editor caret is in. This allows real-time monitoring of how modifying a rule changes the Rete network.

The Jess debugger

The JessDE debugger lets you debug a Jess rules in .clp files, including such standard debugging features as suspend, resume, or step. JessDE debugger also displays the execution stack when the program is halted, and allows examination of variables in the stack frame. Breakpoints can be set or cleared in any on any function (built-in or user defined). Breakpoints are not possible on `defrule` or `deftemplate` construct.

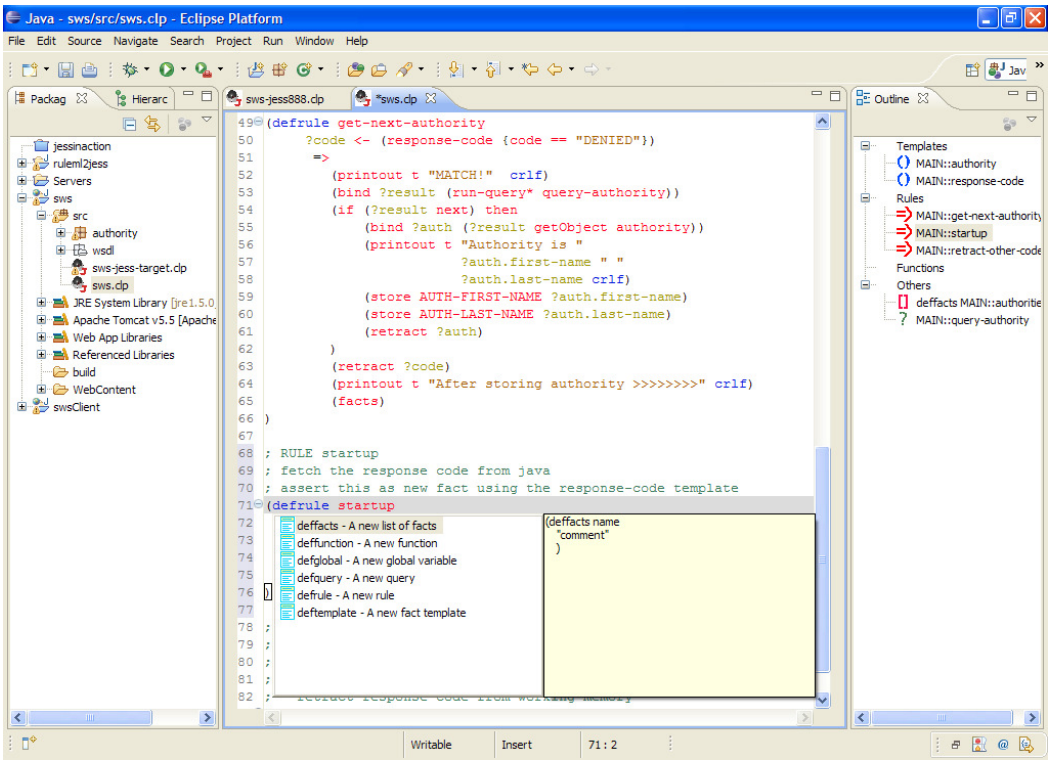


Figure 6-10: JessDE Eclipse Plugin

6.2.5. Deploying Jess rules as a Web service

To allow rules to be incorporated into the design of the Mediator's behavior model in ISDL at the PIM level and invoked by BPEL at the PSM level, we wrap them as Web services using Java. As has been discussed in Section 5.2.4, this is possible since one of the advantages of Jess is its tight integration with the Java programming language. As the Jess rule language and engine are written in pure Java, Jess provides Java APIs that manipulate and execute Jess rules from Java (and vice versa). To Java, Jess rules are mere Java objects, and can simply be invoked using their methods.

To wrap Jess rules in Java as a Web service, we use the Apache AXIS framework. We have used the Eclipse Web Tools Platform (WTP)[55] as the tooling environment to create the necessary Java bean skeletons to wrap Jess rule as a Web service (screenshot shown in Figure 6-12). As with Web service development approaches, there are two ways to achieve this: one is to write a simple Java code with methods that access a Jess rule, and then generate the WSDL specification (bottom up); the other way is to first determine the parameters and return value required by the Jess rule and describe this in a WSDL specification (top down).

Once the WSDL specification is available, we generate the stubs and skeletons of the WSDL and deploy these in Apache Tomcat Servlet as a Web service. The Mediator can then access the Jess rule through the Web service's endpoint. Writing the Java code and WSDL specification, including the deployment and execution of Jess rule as Web service can be facilitated using Eclipse J2EE Platform (in particular the Web Tools Platform (WTP))[56]. Figure 6-11 shows this architecture.

[55] http://www.eclipse.org/webtools/
[56] http://www.eclipse.org/webtools/

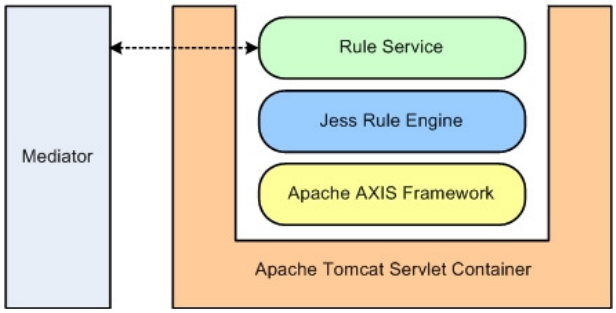


Figure 6-11: Jess rules deployed as web service

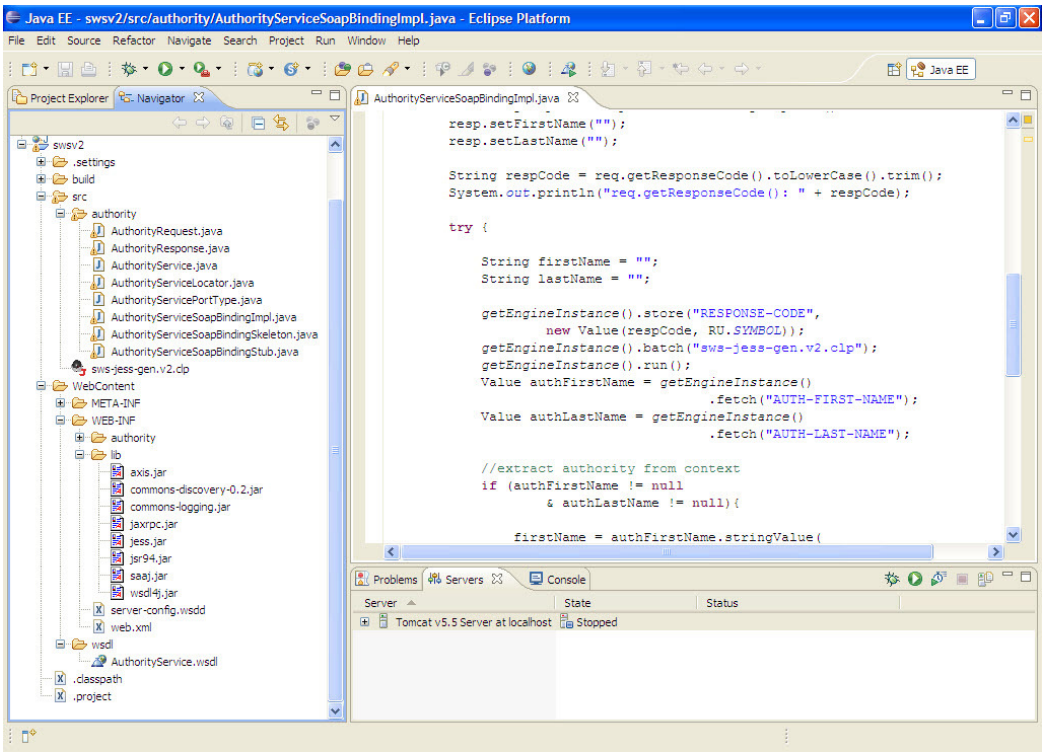


Figure 6-12: Eclipse Web Tool Platform

6.3. Chapter summary

This chapter presented the tools we have used to realize our solution:

Under the *Design Space*, we use BiZZDesign Architect to model architectural components at the business, application, and technology layers in ArchiMate. We use Grizzle to design the behavior model of the mediator in ISDL. Simulation of the ISDL models can be done using Sizzle. We use Tizzle to map the data semantic mismatches between XSD schemata of collaborating systems.

Under the *Goal and Business Rule Space*, we use ARMOR to model goals and relate them to the architectural components using BiZZDesign Architect. The APE Web client gives us an editing environment to test the correctness of our ACE sentences before we translate them into RuleML

transformation using ACE2RRML transformation. Finally, we created a prototype to transform RuleML version 0.9 into Jess based on an earlier similar work.

One observation we can draw from this chapter is that although the technologies by themselves are well supported by tools, there is still an insufficient availability of tools to support the transformation between technologies, especially in the G&BR space. Additionally, all these tools are still isolated from one other. An interesting possible future work would be to design a tooling environment where all these transformations are made available one integrated environment. The Eclipse framework could be a viable solution to this issue.

Part III

Solution Validation

This part applies the goal-driven service mediation solution described in Part II to a case study using the Semantic Web Service Challenge Payment Problem Scenario. We report the results in Chapter 7.

7

SWS Payment Problem Scenario

This chapter illustrates and validates our service mediation solution by applying it to the Payment Problem Scenario[57] provided by the Semantic Web Service (SWS) Challenge[58].

The structure is as follows: Section 7.1 briefly presents what the Semantic Web Service is all about and why we choose this as a way of validating our research. Section describes the scenario in detail. Section 7.3 describes how we apply the methodology step by step. Section 7.4 describes the resulting architecture of the solution in ArchiMate. Section 7.5 compares a solution that does not separate the business logic from the business rules that constrain them. Section 7.6 ends this chapter with a summary.

7.1. The Semantic Web Service (SWS) Challenge

The SWS Challenge is a yearly workshop organized by Stanford University's Logic Group which started in 2006. The goal of the SWS Challenge is to develop a common understanding of various technologies intended to facilitate the automation of mediation, choreography and discovery for Web Services using semantic annotations. The challenge seeks to explore the trade-offs among existing approaches. It also aims to explore which parts of problem space are not yet covered (SWS, 2009).

The Challenge workshops is participated by industry and academic researchers developing software components that have the ability to automate mediation, choreography and discovery processes between Web services. To date, participants of this workshop include Politecnico di Milano/CEFRIEL, DERI (Stanford, Galway, and Innsbruck), Friedrich Schiller University of Jena, Universities of Dortmund and Potsdam, SAP, Fraunhofer FOKUS, University of Georgia/LSDIS, IBM-Max Maximilien, Novay and University of Twente.

Participants who successfully solve the Challenge scenarios get to have their solutions peer reviewed. The workshop also uses evaluation criteria[59] devised by the organizers themselves which mainly assesses the degree of flexibility of presented solutions once changes to the

[57] http://sws-challenge.org/wiki/index.php/Scenario:_Payment_Problem
[58] http://sws-challenge.org/wiki/index.php/Main_Page
[59] http://sws-challenge.org/wiki/index.php/SWS_Challenge_Levels

requirements are made. A higher evaluation success level indicates a better solution to the problem level transition. Finally, solutions which have met the Challenge criteria are certified.

We choose the SWS challenge as a venue to validate our service mediation solution for the following reasons:

- The SWS Challenge Payment Problem scenario focuses on *process mediation* as opposed to previous scenarios where the focus is on *static* data and process mediation (Purchase Order Mediation Scenario[60]) and *dynamic* data and process mediation (Purchase Order Mediation Scenario v.2[61]). In particular, the Payment Problem Scenario is designed to seek solutions that overcome problems of *goal-based* Web service compositions using Web technologies. It also focuses on designing process mismatches whose business logic needs to be taken care of by the Mediator. The nature of the challenge is thus opportune and most appropriate to the theme of this thesis.
- Although the Payment Problem Scenario presents only a fictional set of problems that tackle issues about interoperability between existing heterogeneous systems; however, it does derive these problems from real industry situations which are generic enough to be applied to any industry. It is regarded by participating industry and research researches as that which provides a sufficient degree of rigor and complexity and a common ground by which they can adequately apply, validate, and compare their solutions.
- As the SWS Challenge invites a number of practitioners and researchers to solve a common problem, it also provides the opportunity for their solutions to be peer reviewed by other researchers working in the area of service mediation. We would like to similarly take advantage of this situation by presenting our solution and be reviewed by others.
- An attractive feature of the Challenge is that it provides a test bed facility where we can simulate the solution with actual Web service invocations. The Challenge also provides a set of criteria that must be satisfied in order for the solution to be certified thus providing us an additional opportunity for validation.
- Finally, we choose this scenario as a way of contributing to the continued scientific work on service mediation through the SWS Challenge.

7.2. The Payment Problem Scenario

We now describe the SWS Challenge Payment Problem Scenario in detail. *Blue*, a fictional company, wants to integrate functionalities of its two internal Accounting and Management Department Systems (ADS and MDS, respectively) with the external Financial Information Provider (FIP) of *Moon* – another fictional company. Blue's Mediator, shown in Figure 7-1, is tasked to handle the logic required in handling payments on purchased orders (described in the previous scenarios). For goal identification purposes, we first state verbatim the general case description provided by the SWS website. We then later describe the required steps in detail based on our understanding of the general case description.

General case description

> The Payment Problem represents an extension of Purchase Order Mediation v2 scenario with an example of purchase order payment initiation procedure. The aim of this scenario is to show how semantic Web technologies can contribute in overcoming problems of goal-based web service compositions.

[60] http://sws-challenge.org/wiki/index.php/Scenario:_Purchase_Order_Mediation
[61] http://sws-challenge.org/wiki/index.php/Scenario:_Purchase_Order_Mediation_v2

After receiving an ACK of the purchase order Blue's goal is to initiate purchase order payment by sending a message in the form of *UNIFI ISO 20022 Payments Standard Initiation - Customer Credit Transfer Initiation V02* (payment instruction) to its Financial Department (FD). The payment instruction must be compiled with all necessary data from various sources (Blue's and Moon's addresses, bank accounts and bank identifiers, PO amount, etc). Under certain circumstances payment instruction must be authorized by Blue's Management Department (MD). Upon payment instruction dispatch Blue expects to receive from its FD *UNIFI ISO 20022 Payments Standard Initiation - Payment Status Report V02 message* (payment status report).

There are three steps that must be accomplished in order to compile and send a payment instruction:

1. Moon's banking information must be retrieved (unconditional step). For this step Moon's Financial Information Provider service is used.
2. The payment should be authorized by Blue's authorization system (conditional step). For this step Blue's Management Department service is used.
3. The payment must be processed by Blue's payment system (unconditional step). For this step Blue's Financial Department service is used.

Blue initiates request for payment of orders by sending the PaymentInitiation message formatted according to *UNIFI ISO 20022 Payments Standard Initiation - Customer Credit Transfer Initiation* (CCTI) *V02* – a payment standard in the financial industry. This message basically contains, among others, debtor and creditor information, transaction information (e.g. ID, date, account number, currency, amount, etc.), means of payment (e.g. credit card, check), etc.

Blue has an internal policy that payments under 2000 Euro can be authorized automatically by using only the Financial Department service (in that case second step could be removed). For payments over 2000 Euro, Financial Department service will reject payment instruction, unless accompanied by an otherwise optional authorization code, which can only be obtained by making a request to the Management Department service. The inputs to the Management Department service are same as those to the Financial Department service, except that an Authority must be designated. This service returns either an authorization code or a denial code. If denial code is returned, the service may be called again, but not with the same Authority as in previous call. The Authority can be any Blue employer by with proper legitimacy, based on the amount of money up to which an authorization can be given.

For example, Jackie Brown can authorize the amounts up to 2000 Euro, and Cathy Johnson up to 3000 Euro. Furthermore, Blue has a policy that the least senior Authority, as determined by increasing amount of money up to which an authorization could be made, should be requested first. The complete list of Authorities and designated amounts can be found in Table 7-1.

List 7-1: SWS Challenge Payment Problem Scenario case description

Order of Invocation	Authority	Designated Amount (maximum)
1st	Jackie Brown	2 000
2nd	Cathy Johnson	3 000
3rd	Arnold Black	10 000
4th	Peter Petrelli	50 000

Table 7-1: Authorities and their designated amounts

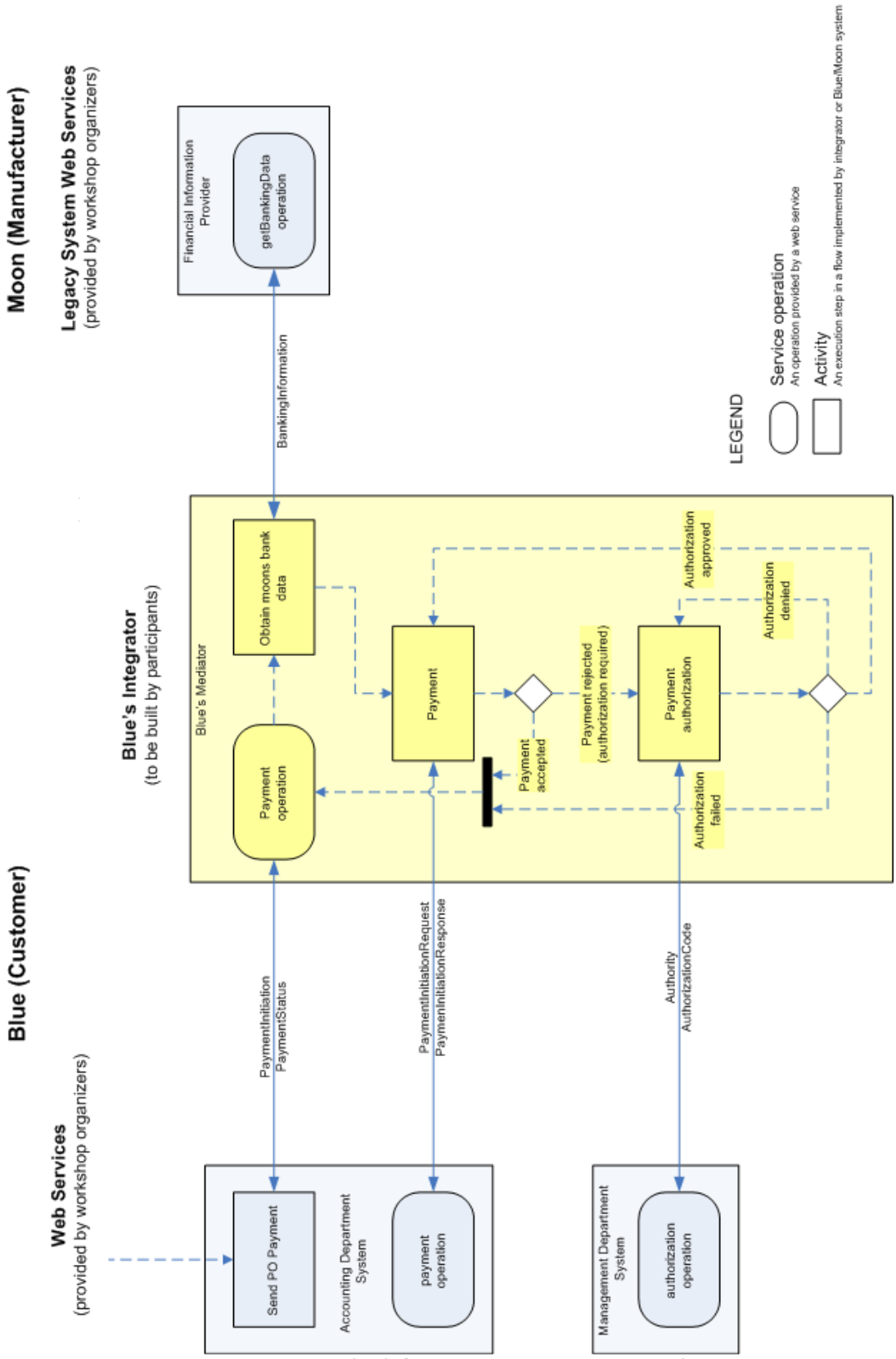


Figure 7-1: SWS Challenge Payment Problem Scenario

Detailed case description

Blue uses the `initiatePayment` operation to invoke and trigger the Mediator service (to be developed in this case). Upon receipt of `PaymentInitiation` message which is formatted according the *UNIFI ISO 20022 Payments Standard Initiation - Customer Credit Transfer Initiation V02*, the Mediator retrieves Moon's banking information from its FIP using the `getBankData` operation. Moon sends a short reply back to the Mediator containing the Blue's `swiftCode`, `accountNumber`, and `currencyCode` as `BankingInformationResponse` which is formatted according to Blue's own standard.

Together, the `PaymentInitiation` and `BankingInformationResponse` messages, and their mismatches, are combined to form the complete `PaymentInitiationRequest`. This message is passed using the `processPayment` operation to determine if the purchase `amount` can be processed or a designated Authority officer is needed by Blue's ADS. The ADS can either return a `PROCESSED` response automatically when the amount is below `€2000`; otherwise, an `AUTHREQUIRED` response is returned.

When the response is `PROCESSED`, the Mediator proceeds to create the `PaymentStatus` message formatted according to *UNIFI ISO 20022 Payments Standard Initiation - Payment Status Report* (PSR) *V02*, and attaches `PI_ACCEPTED` as the `paymentStatusCode`. The `PaymentStatus` serves as the response message to the `initiatePayment` operation.

When the response is `AUTHREQUIRED`, the Mediator needs to make subsequent calls to Blue's MDS using the `authorize` operation. For each call, the Mediator inserts the Authority Officer's name in the `AuthorizationRequest` message (which is basically the same in structure with `PaymentInitiationRequest` only with an `Authority` element).

Each Authority Officer of Blue's MDS has a designated amount to which he/she can authorize a payment. For example, `Jackie Brown` can only authorize payments from `€0` to `€2000`. Authorities are used by the Mediator in an ascending order; that is, if for example, the purchase `amount` is `€3500`, the Mediator sends `Jackie Brown` as the Authority first. Since, she is not authorized with this amount, the Mediator invokes the `authorize` operation again, this time, with `Cathy Johnson` as the next higher Authority.

After every invocation of the `authorize` operation, Blue's MDS replies either `DENIED` or `ACCEPTED`. The response is `DENIED` when the purchase amount cannot be authorized by the current Authority. The Mediator continues this invocation until an appropriate Authority can authorize the payment. If all Authorities have been exhausted, the Mediator creates a `FAILED` response. On the other hand, the response is `ACCEPTED` when there is an Authority that can authorize the amount. Along with the `ACCEPTED` response, an `authorizationCode` is also sent back to the Mediator which is used to invoke the `processPayment` the second time. The ADS *should* return `PROCESSED` afterwards.

Finally, when the response code is `PROCEESED`, the Mediator creates a reply to the `initiatePayment` invoked at the outset. The reply message `PaymentStatus is` formatted according to PSR whose contents are derived from CCTI. This message is appended with a `paymentStatusCode` which describes the final result of the payment initiation request: `PI_ACCEPTED` when Blue accepted the payment or `PI_REFUSED_AUTH_FAILED` when request for payment and authorization has failed.

Figure 7-2 shows the activity diagram describing the interactions between the operations of Blue and Moon, including the business logic that needs to be handled by the Mediator.

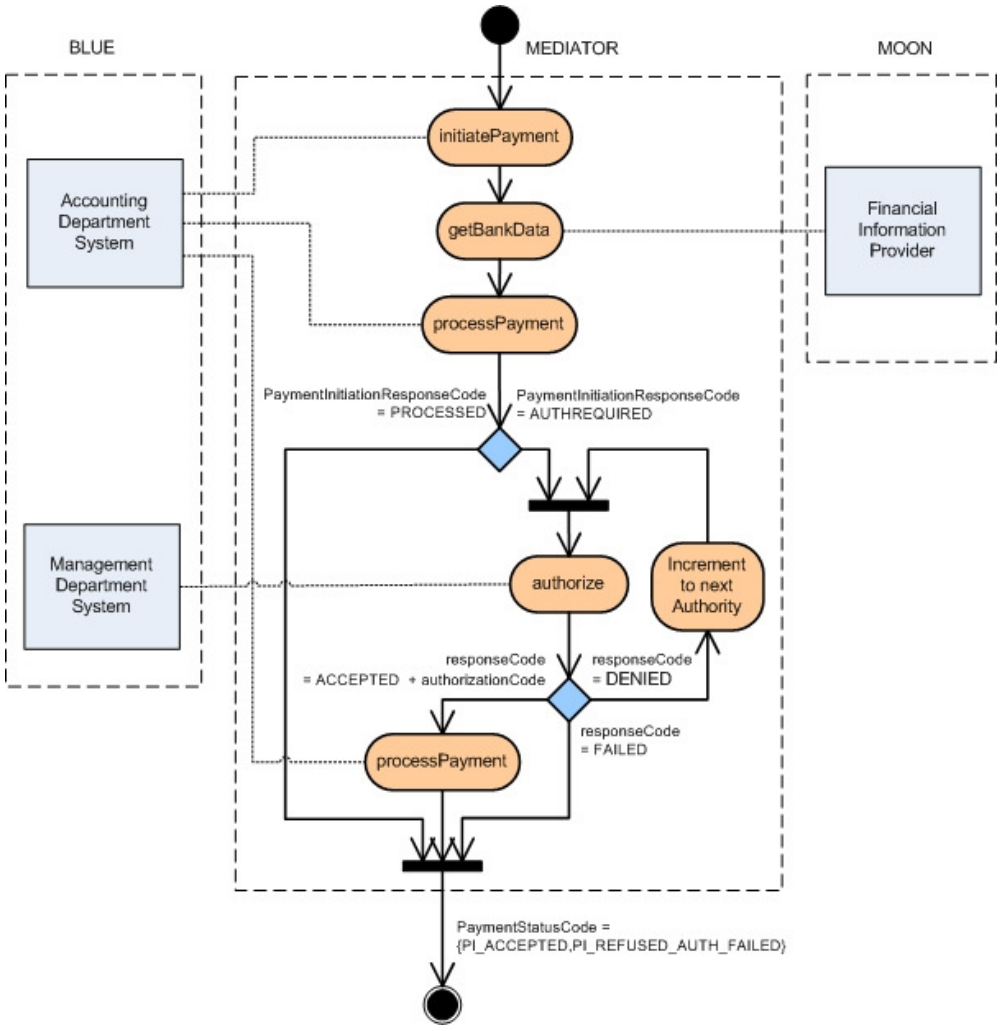


Figure 7-2: Payment Problem Scenario activity diagram

7.3. Applying the methodology

We now solve the SWS Challenge Payment Problem Scenario according to the service mediation solution described in Chapter 5. We go through each step of the methodology along with the tools that will be used.

Scope

Due to time constraints, we will not perform the last step of the methodology which is to transform the generated ISDL models into their equivalent representations in BPEL as described previously in Step 7 of Section 5.4.7. Also as our validation, we only perform simulation in ISDL.

7.3.1. Step 1 – Abstracting from PSMs to PIMs

We first look at the existing services offered by the different applications of Blue and Moon through their WSDL descriptions, Table 7-2 shows a summary of each application's services,

their operations, and the parameters of their messages. For more details of the WSDL specifications, please see the SWS Challenge Payment Problem Scenario website[62].

Applications	WSDL Document	Services	Operations	Messages	
Blue's Accounting Department System (ADS)	FDService.wsdl	Financial Department Payment Service	process Payment	in	Payment Initiation Request
				out	Payment Initiation Response
Blue's Management Department System (MDS)	MDService.wsdl	Management Department Payment Service	authorize	in	Authorization Request
				out	Authorization Response
Moon's Financial Information Provider (FIP)	FIPService .wsdl	FIPService	getBanking Data	in	Banking Information Request
				out	Banking Information Response
Blue's Integrator (Mediator)	PaymentService .wsdl	Payment Service	initiate Payment	in	Payment Initiation Request
				out	Payment Initiation Response

Table 7-2: Summary of WSDL operations and messages

The abstraction process involves taking the Web service operations and representing them as either *operation calls* (client side) or *operation execution* (server side) in ISDL. At the same time, we are assigning them roles in their interactions without without having to directly handle the WSDL documents themselves. For example in Figure 7-3, the `initiatePayment` operation will be represented as an *operation call* on Blue's side as Blue acts as a client to the Mediator. We model each application of Blue and Moon (i.e. ADS, MDS, and FIP) as distinct behaviors in ISDL shown in Figure 7-3. This step can be automated through the import functionality of Grizzle.

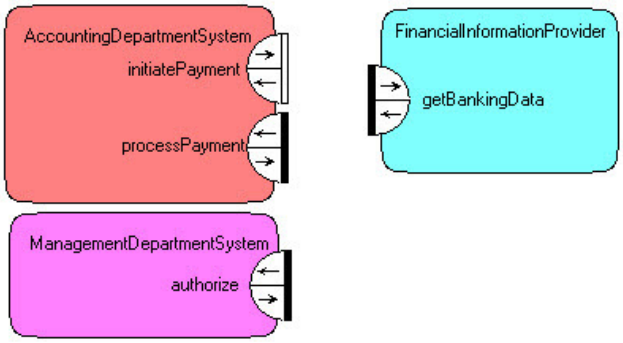


Figure 7-3: Generated ISDL model for Blue and Moon

The information model derived from the WSDL operation's message parameters (shown in Table 7-2) needs to be represented as Java classes (for execution) and UML (for visualization). Transforming the XSD schema of the WSDL messages can be done using Tizzle. Visualizing the Java classes as UML models can be done using Omondo EclipseUML's reverse engineering feature.

[62] http://sws-challenge.org/wiki/index.php/Scenario:_Payment_Problem

For example, Blue's `initiatePayment` operation uses the `PaymentInitiation` message that is formatted according to the *UNIFI ISO 20022 Payments Standard Initiation - Customer Credit Transfer Initiation V02* XSD schema to send payment information to the Mediator. Part of the message schema is the `CdtTrfTxInf` message element which is composed of a set of sub-elements bearing transaction information such as `Amt` (i.e., amount of money to be moved between the debtor and creditor, before deduction of charges), `BIC` (i.e., the Bank the Identifier Code as a sub-element of `CdtrAgt`), `IBAN` (i.e., the International Bank Account Number as a sub-element of CdtrAgtAcct), and `Ccy` (i.e., currency as an sub-element of `CdtrAgtAcct`). The generated JAXB code from the XSD schema of `CdtTrfTxInf` is partially shown `in` List 7-2. The UML diagram and other associated classes are shown in Figure 7-4.

```
470 @XmlAccessorType(XmlAccessType.FIELD)
471 @XmlType(name = "", propOrder = {
472     "amt",
473     "cdtrAgt",
474     "cdtrAgtAcct",
475     "cdtr"
476 })
477 public static class CdtTrfTxInf {
478
479     @XmlElement(
480             name = "Amt",
481             namespace = "http://www.sws-challenge.org/schemas/iso20022",
482             required = true)
483     protected Pain00100102 .PmtInf.CdtTrfTxInf.Amt amt;
484     @XmlElement(
485             name = "CdtrAgt",
486             namespace = "http://www.sws-challenge.org/schemas/iso20022",
487             required = true)
488     protected Pain00100102 .PmtInf.CdtTrfTxInf.CdtrAgt cdtrAgt;
489     @XmlElement(
490             name = "CdtrAgtAcct",
491             namespace = "http://www.sws-challenge.org/schemas/iso20022",
492             required = true)
493     protected CashAccountType cdtrAgtAcct;
494     @XmlElement(
495             name = "Cdtr",
496             namespace = "http://www.sws-challenge.org/schemas/iso20022",
497             required = true)
498     protected PartyIdentificationType cdtr;
```

List 7-2: Generated JAXB code fragment of XSD element `CdtrTrfTxInf`

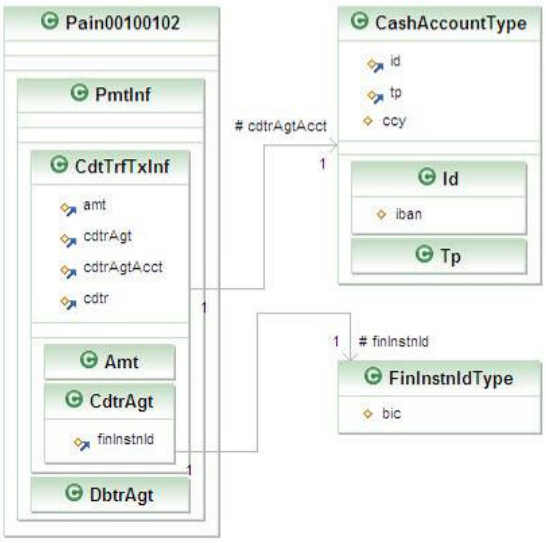


Figure 7-4: UML diagram of `CdtrTrfTxInf`

Furthermore, the information model of the WSDL operation's message parameters will be used in ISDL to represent the operation's *information type* – depicted as a rectangle with a line connecting the operation. The information type shows the operation's input and output messages derived from its WSDL specification. For example, in Blue's `initiatePayment` operation, the `PaymentInitiation` (declared with a variable `initPay`) input message will be represented as an `Accept` parameter, while the `PaymentStatus` output message will be represented as a `Reply` parameter (declared with a variable name `statPay`). This is shown in Figure 7-5.

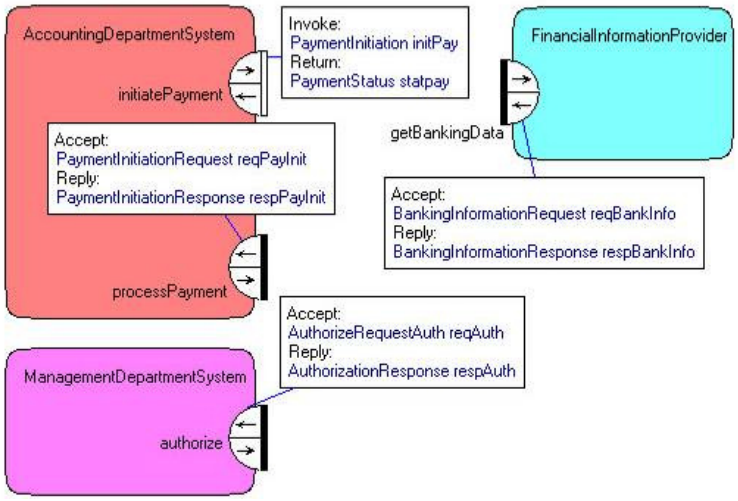


Figure 7-5: Generated ISDL model for Blue and Moon with information type

7.3.2. Step 2 – Semantic enrichment of PIMs

Semantically enriching the behavior model of either Blue or Moon is quite easy seeing that each behavior has at the most two operations. From Blue's Accouting Department System, we know from the case description of the scenario that `initiatePayment` should be executed first before `processPayment`. We represent this ordering simply using ISDL's *enabling condition* construct (i.e. arrow) that connects them both shown in

Figure **7-6.**

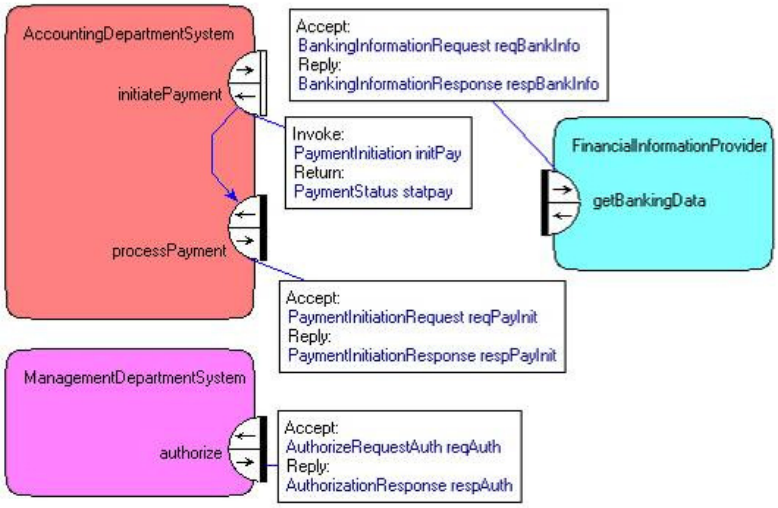


Figure 7-6: Semantically enriched ISDL behavior model

To semantically enrich the information models, we need to relate or "match" the meanings (i.e. semantics) between Blue's and Moon's messages. We can use the generated UML diagram as a guide to do this. Currently, determining which message elements match other elements is done manually. In real cases, the meanings between these messages can be best determined by business domain experts of the collaborating enterprises. Figure 7-7 shows an example of how Blue's `CdtrTrfTxInf` message elements match to those of Moon's `BankingInformationResponse` message elements.

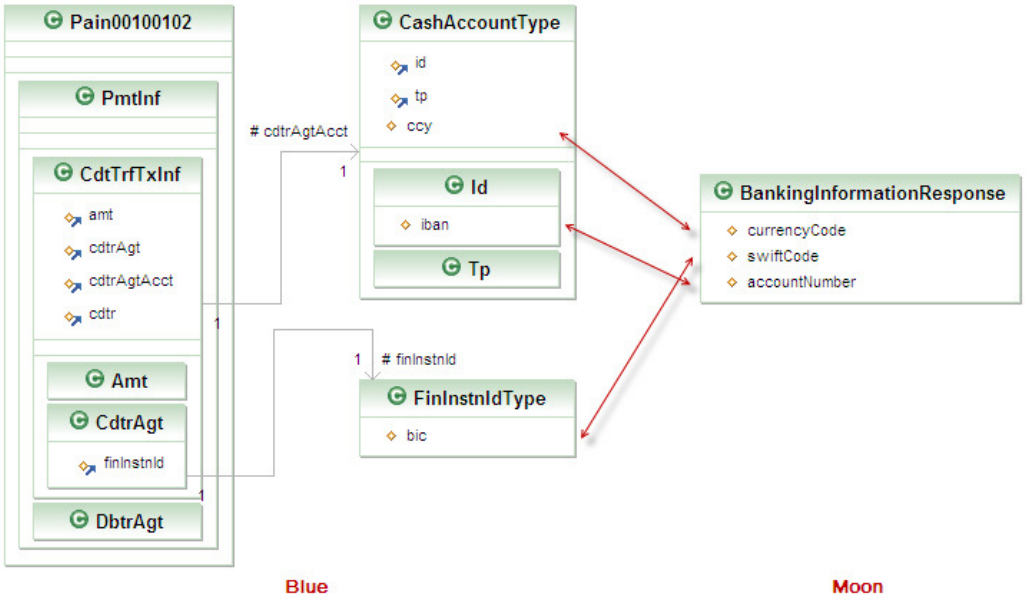


Figure 7-7: Semantically enriching information model between Blue and Moon

At this stage, we have determined which services already exist as can be derived from the WSDL specifications of Blue and Moon. We then represented them as operations calls or operation executions in ISDL. We have also determined how operations within Blue and Moon are ordered (not between them yet). We have also related the message elements by looking at their UML models (i.e., how one message element of Blue semantically matches to those of Moon).

This essentially completes our bottom-up analysis of available resources; the next step is to specify the requirements of the integration using goal modeling in ARMOR and match them to these resources.

7.3.3. Step 3 – Model goals and business rules at CIM

Substep 1: Model goals in ARMOR

Identification of stakeholders and the primary goal of integration

Using goal identification techniques (which in our case is by searching for intentional keywords from the case description described in Section □), we find that the primary goal of the integration is "`to initiate purchase order payment`". We provide a name to this goal as "`Pay purchase order`". We depict it as a *hard goal* in ARMOR seeing that the interactions of the messages provide us a way of determining the result concretely. Furthermore, it is evident that the stakeholders of the integration are Blue, Moon and Mediator. Our preliminary goal model in ARMOR is shown in Figure 7-8.

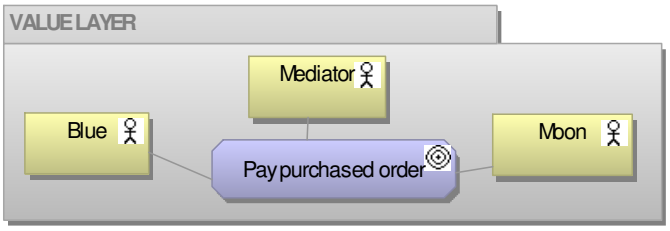


Figure 7-8: Stakeholder and primary goal model in ARMOR

Refinement of primary goals

In refining the hard goal of the integration, we choose to represent the sub-goals as use cases in ARMOR. Use cases allow us to describe who the actors of the integration are, including the goals by which these actors need to perform to contribute to the satisfaction of the main goal (which in our case is the hard goal of paying purchased order), without first providing details as to how these goals are to be implemented by the underlying systems.

Looking again at the case description in Section 7.2, we know that Blue's goal is to perform the actual processing of payments (depicted in Figure 7-9 as a use case with the name "`Process payment`"), Moon's goal is to provide banking data (depicted in Figure 7-9 as a use case with the name "`Provide banking data`"), and finally, the goal of the Mediator is to consolidate messages and operations coming from both Blue and Moon in order to make a correct request for payment (depicted in Figure 7-9 as a use case with the name "`Consolidate data`"). We use the `AND`-realization construct to depict the idea that when all of these use cases are satisfied, the main hard goal will be also be satisfied.

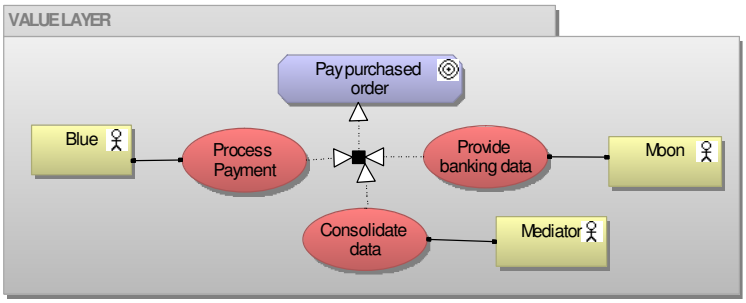


Figure 7-9: Refining primary goal into high-level use cases

Our next step is to further refine the use case in Figure 7-9 to a point where it can be expressed as a requirement in ARMOR. In the process of refinement, we can use the goal refinement/decomposition technique described in Section 3.4.2. Figure 7-10 shows the refinement of each use case. For example, the use case "`Process payment`" can be refined further by asking *how* the processing of payment should take place. Looking at the case description in Section 7.2, we can see that Blue accomplishes the processing of payment by requiring authorization for amount greater than €2000. We depict this as a child use case named "`Authorize payment`". The use case relation `<<include>>` relates the parent use case and other child use cases. This depicts the idea that the parent use case will be satisfied when the sequence of interactions between the child use cases are accomplished.

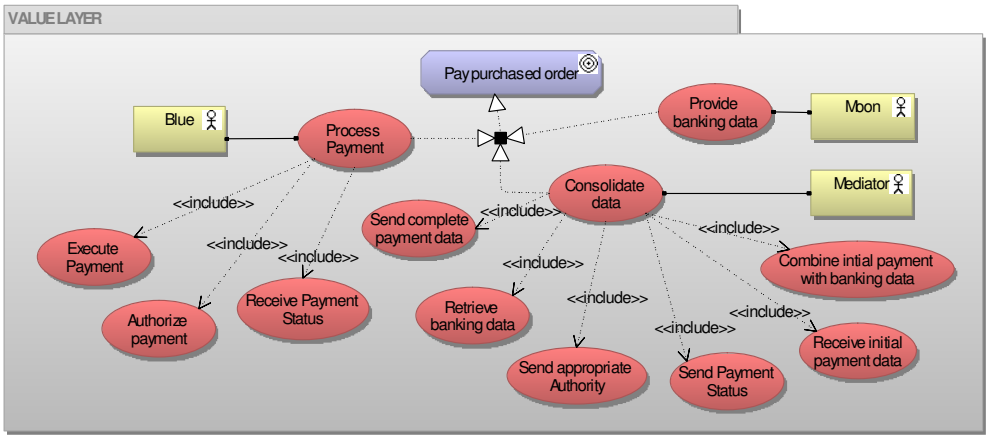


Figure 7-10: Refined primary goal using use cases

Refinement of sub-goals into requirements

At this stage, we have refined the parent use cases into child use cases. Our next objective is to refine the child use cases into *requirements*. ARMOR defines a requirement as a *goal* that can be assigned to a system such that the system is made responsible for the satisfaction of the goal. This means that the system, whose functionalities are invoked to achieve the requirement, satisfies the child use case, in turn the parent use case, and ultimately the parent hard goal. Refining a requirement from a use case is possible as can be seen from ARMOR's abstract syntax shown in Figure 5-1.

Figure 7-11 shows requirements (depicted as rectangles with angled corners colored blue-green) realizing one or more child use cases. For example, the "`Authorize payment`" requirement realizes Blue's child use case "`Authorize payment`".

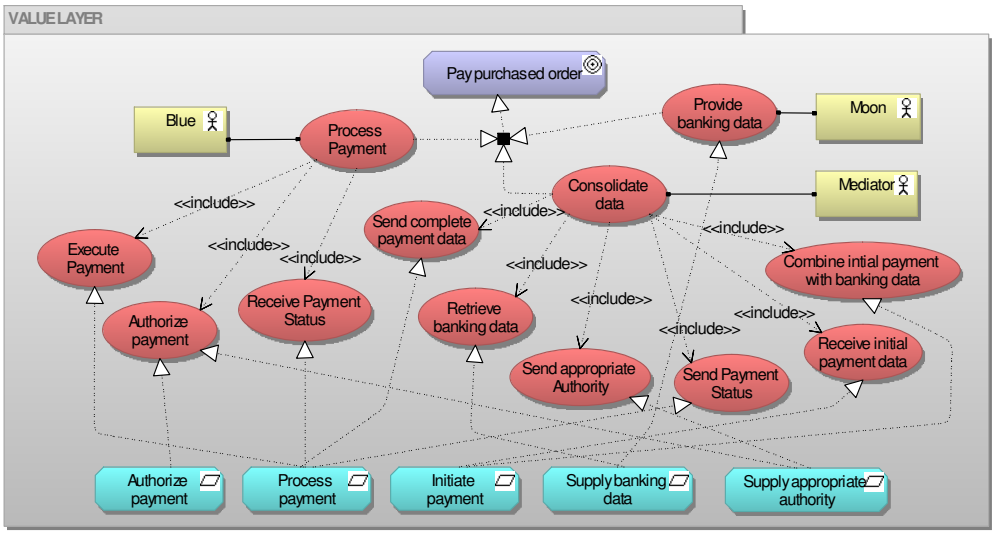


Figure 7-11: Refining use cases as requirements

Substep 2: Map requirements to an existing service in ARMOR

Seeing that our requirements have been defined, we now turn to the existing services of both Blue and Moon that were earlier derived from Steps 1 and 2. Our objective is to determine which existing service can be assigned to a requirement.

Figure 7-12 shows how we mapped the requirements with the existing services using the *realization* construct of ArchiMate (depicted dashed line with a hollow arrowhead). In essence, ArchiMate defines the realization construct as a manner of linking a logical entity (in this case, the requirement) with a more concrete entity (in this case, the service) that realizes it. For example, the Management Department Service (c.f. Table 7-2) when invoked satisfies the requirement "`Authorize payment`".

Notice that through reading the case description of the scenario, we are not able to find any service that satisfies the requirement "`Supply appropriate Authority`".

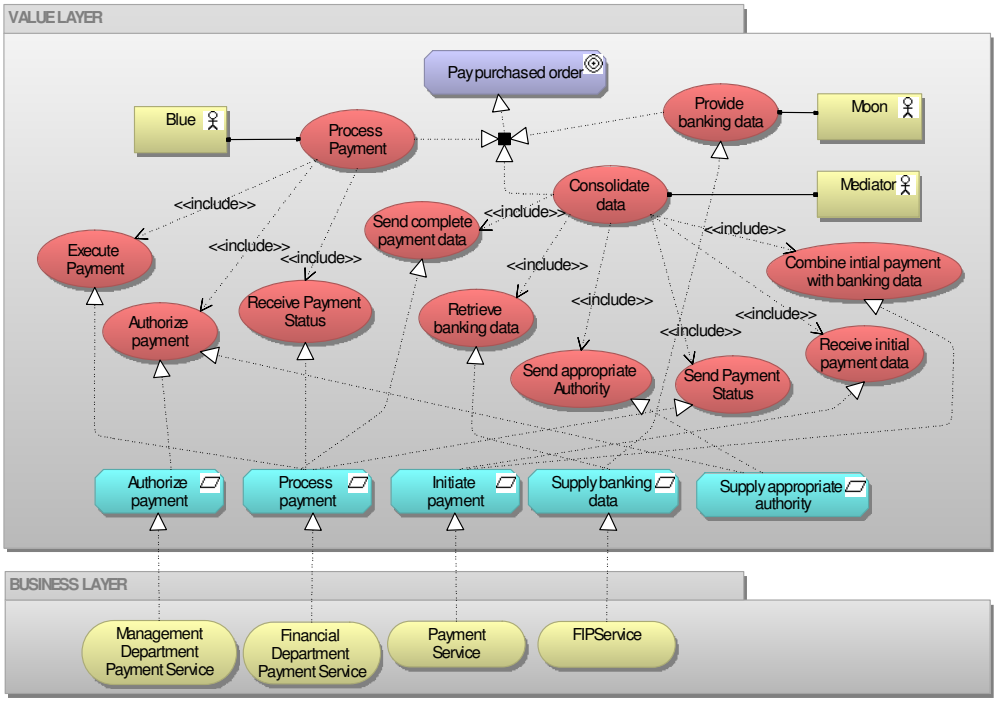


Figure 7-12: Mapping requirements to existing services

Substep 3: Refine requirement as business rule

Figure 7-12 shows that the requirement "`Supply appropriate Authority`" cannot be realized by any service derived from Steps 1 and 2. Reading the case description in Section 7.2, however, tells us that an appropriate Authority should be sent to Blue when an amount is greater than €2000. This first and last name of the Authority, with a certain designated amount (cf. Table 7-1) should be sent with subsequent invocation to the `processPayment` operation of the Moon's Financial Department Service. We can represent this as a business rule as it essentially constrains the business process. We thus represent this using the business rule construct in ARMOR. Refining requirements as a business rule in ARMOR is possible as can be seen from ARMOR's abstract syntax shown in Figure 5-1.

Figure 7-13 shows how we refined the requirement into a business rule whose description is stated in plain English. Since no service current exists to realize this requirement, we need to make this business rule executable, expose it as a service, and incorporate it in the design of the behavior model of the Mediator ISDL.

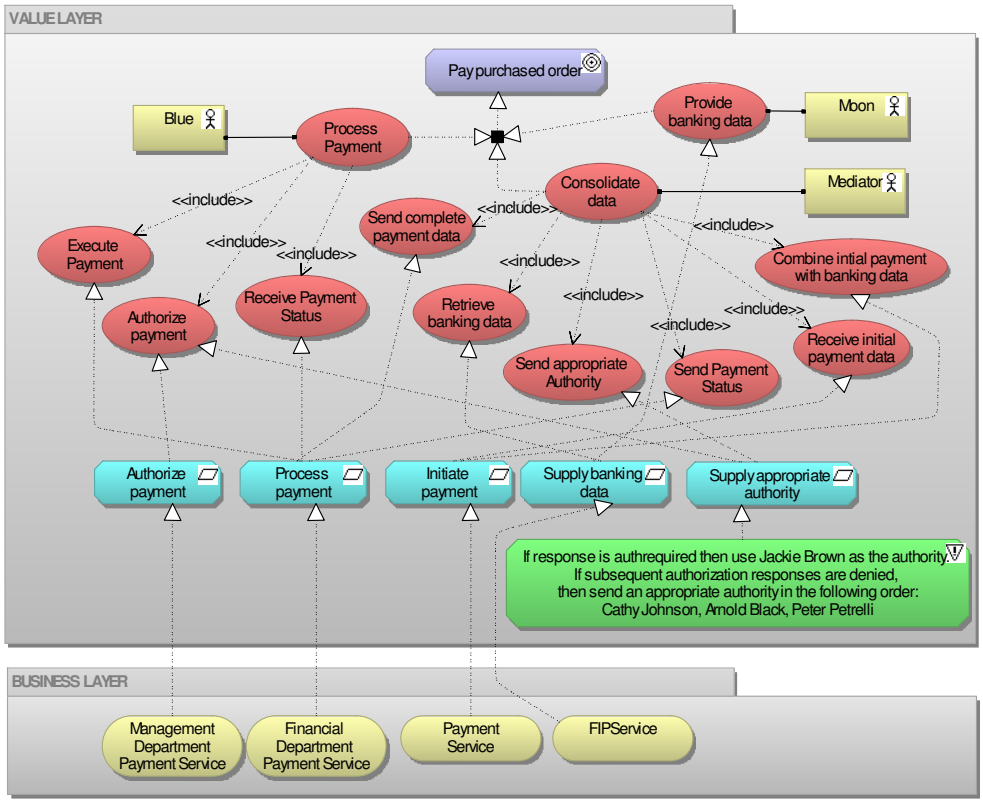


Figure 7-13: Refining a requirement as a business rule

Substep 5: Translate business rules into ACE

Figure 7-14 shows again the business rule depicted in Figure 7-13 for emphasis. As we have mentioned in Section 3.5, one of the properties of a business rule is that, aside from the fact that they can be represented in near-natural language, they can also be made executable. Our objective in this step is thus to translate the business rule modeled in ARMOR to ACE.

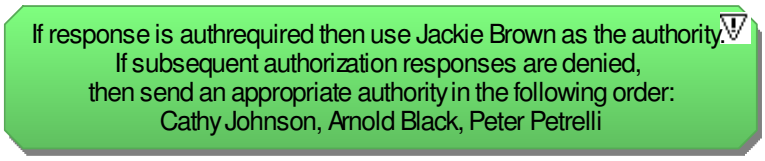


Figure 7-14: Business rule modeled in ARMOR

Translating business rules from plain English to ACE is done manually with the assistance of the APE Web client (cf. Section 6.2.1). Our approach has been to divide the business rules modeled in ARMOR into four detailed rules in ACE. Following the recommended steps in using ACE (cf. Section 6.2.1), we first state the rules using `IF…THEN` statements and arrived at an initial ACE translation. The first rule has thus been stated as:

```
If the response is authrequired
then the next authority is Jackie Brown.
```

However, our initial try gives us an error. APE does not have the words `authrequired`, `next`, `authority`, and `Jackie Brown` as part of its lexicon. Thus we need to introduce some word-class markers to tell APE how we interpret the words (cf. Section 5.2.2). We prefix these words with "`a`" for adjective, "`n`" for noun, and "`p`" for proper noun depending on how we used them. Furthermore, compound words not recognized by the APE parser

must be connected with a dash; APE then treats this as a proper noun. Thus, we have the following rule:

```
If the n:authrequired is a:denied
then the a:next n:authority is p:Jackie-Brown.
```

APE now recognizes the rule as a valid ACE sentence as shown in Figure 7-15. We repeat this process for the rest of the business rules in ARMOR.

The four ACE translation of the business rules are shown in List 7-3. The next step is to have the rules validated again by the business domain experts. We may need to take out the world-class markers to simplify the ACE sentences before showing them to the business domain experts for validation.

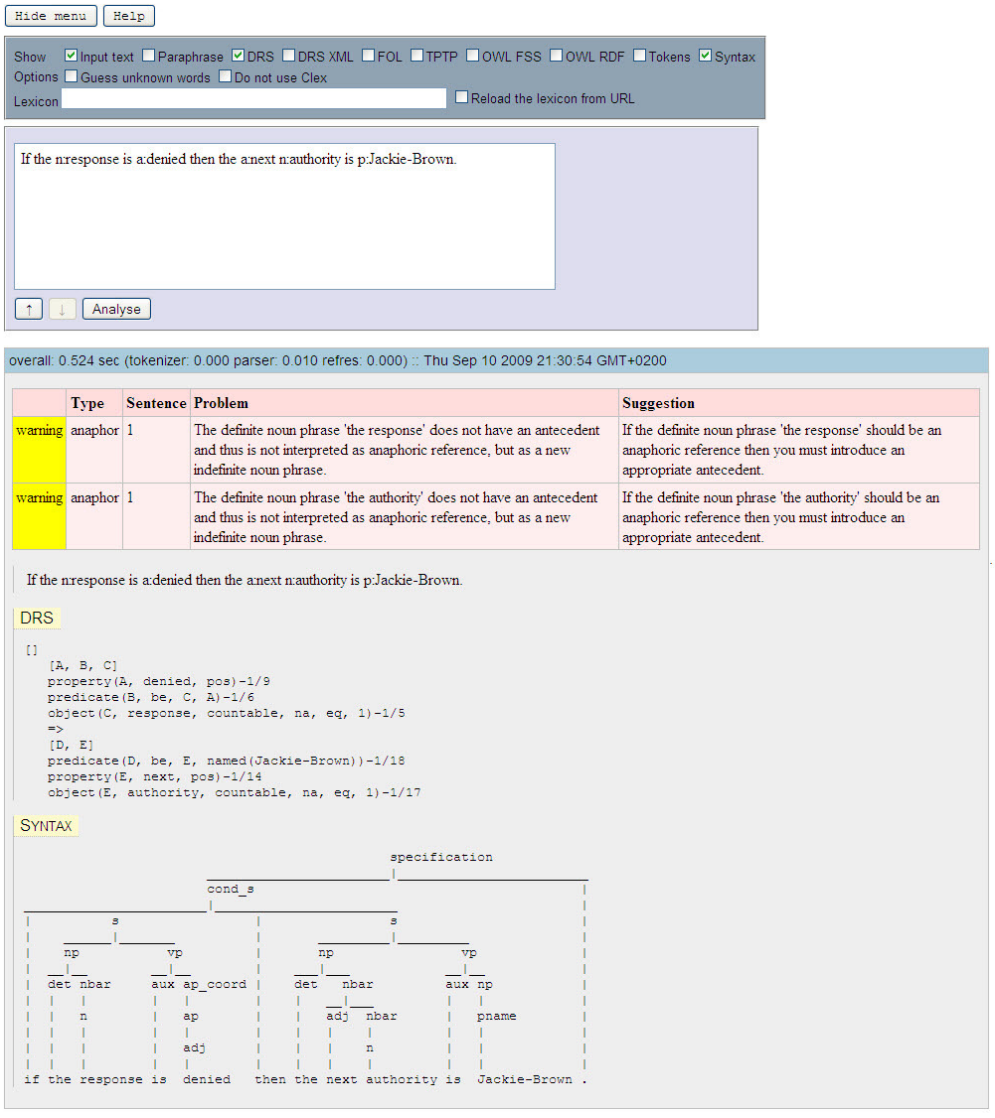


Figure 7-15: Translating business rules in APE

Note that from the equivalent DRS representation of the ACE sentence in Figure 7-15, APE translates all nouns or subjects of the subordinate clauses as an object, adjective complements as a property, and linking verbs as predicates. Also, APE represents the proper noun `Jackie Brown` as "`named(Jackie-Brown)`".

```
Authority1:

If the response is authrequired
then the next authority is Jackie-Brown.


Authority2:

If the response is denied
and the previous authority is Jackie-Brown
then the next authority is Cathy-Johnson.


Authority3:

If the response is denied
and the previous authority is Cathy-Johnson
then the next authority is Arnold-Black.


Authority4:

If the response is denied
and the previous authority is Arnold-Black
then the next authority is Peter-Petrelli.
```

List 7-3: Translated business rules to ACE

7.3.4. Step 4 – Transformation of business rules

Substep 1: Transformation of ACE to RuleML.

We use the ACE2RRML Web client to translate ACE to RuleML. As ACE2RRML uses APE to parse ACE sentences, we need to use the ACE sentences with their appropriate word-class markers as input to ACE2RRML. Additionally, we give the name `Authority1` as the `conversation id` which will be later transformed as the name of the rule in Jess. We do this step for each rule stated in List 7-3. The resulting RuleML fragment translation for `Authority1` is shown in List 7-5. For a complete list of all the RuleML translations, please see Appendix A.

Substep 2: Transformation of RuleML to Jess Rule Language.

We created a prototype for transforming RuleML to Jess using XSL Transformation. Executing the XSLT stylesheet is done simply by commercial tools such as Oxygen XML editor or by writing a simple Java code. List 7-4 shows the equivalent Jess rule translation of the RuleML specified in List 7-3. After all the Jess rules have been translated, we put them into one batch file with an extension "`.clp`" which in our case is called "`sws-jess-gen.v2.clp`".

For the complete XSLT stylesheet and the complete list of all translated Jess rules, please see Appendix A. For more details about the RuleML2Jess translation, please Appendix B.

```
1 (defrule Authority1
2     (response authrequired)
3         =>
4         (assert (authority Jackie Brown)
5     ))
```

List 7-4: Translated Jess rule (Authority1)

```
<Forall>
    <Var>A</Var>
    <Var>B</Var>
    <Var>C</Var>
    <Implies>
        <And>
            <Atom>
                <Rel>property</Rel>
                <Var>A</Var>
                <Ind>authrequired</Ind>
                <Ind>pos</Ind>
            </Atom>
            <Atom>
                <Rel>predicate</Rel>
                <Var>B</Var>
                <Ind>be</Ind>
                <Var>C</Var>
                <Var>A</Var>
            </Atom>
            <Atom>
                <Rel>object</Rel>
                <Var>C</Var>
                <Ind>response</Ind>
                <Ind>countable</Ind>
                <Ind>na</Ind>
                <Ind>eq</Ind>
                <Data>1</Data>
            </Atom>
        </And>
        <Exists>
            <Var>D</Var>
            <Var>E</Var>
            <And>
                <Atom>
                    <Rel>predicate</Rel>
                    <Var>D</Var>
                    <Ind>be</Ind>
                    <Var>E</Var>
                    <Var>named('Jackie-Brown')</Var>
                </Atom>
                <Atom>
                    <Rel>property</Rel>
                    <Var>G</Var>
                    <Ind>next</Ind>
                    <Ind>pos</Ind>
                </Atom>
                <Atom>
                    <Rel>object</Rel>
                    <Var>E</Var>
                    <Ind>authority</Ind>
                    <Ind>countable</Ind>
                    <Ind>na</Ind>
                    <Ind>eq</Ind>
                    <Data>1</Data>
                </Atom>
            </And>
        </Exists>
    </Implies>
</Forall>
```

List 7-5: Translated ACE rule (Authority1) to RuleML

Substep 3: Deployment of Jess rules as web services.

Our next task is to expose the Jess rules as a Web service using Java. As we have mentioned in Section 5.2.4, one of the most powerful features of Jess is its tight integration with the Java programming language. Jess provides some mechanisms to allow Java to invoke Jess rules. We first describe the process of deploying a Web service that will wrap the Jess rules as a Web service. We then describe the necessary codes to connect Java and Jess.

To wrap the Jess rules as a Web service, we create a WSDL specification that has one operation and two message parameters. We call our Web service as `AuthorityService`. It has one operation called `getNextAuthority`. It takes an input parameter called `AuthorityRequest` message that contains the `responseCode` returned by the `paymentInitiation` operation (cf. Table 7-1) of Moon's Financial Information Provider Service whose values could either be `AUTHREQUIRED`, `DENIED`, or `FAILED`. As its output parameter, it uses the `AuthorityResponse` message which contains the Authority's first and last names. Table 7-3 provides a summary.

Applications	WSDL Document	Services	Operations	Messages	
JessRule Engine	Authority .wsdl	Authority Service	getNext Authority	in	Authority Request
				out	Authority Response

Table 7-3: Summary of operations and message of Authority Service

Our next step is to generate Java bean skeletons using the `AuthorityService.wsdl`. This is accomplished using Eclipse WTP framework. From the generated Java skeletons, we modify the `AuthorityServiceSoapBindingImpl.java` class which has the `getNextAuthority` method shown partially in List 7-6. This method will be invoked by the AXIS framework to execute the Jess rule.

In particular, the `store()` method of the Jess engine takes the response code from Java and stores it in a Jess variable called `RESPONSE-CODE`. The `batch()` parses the "`sws-jess-gen.v2.clp`" file as a Jess code. The `run()` method starts the Jess running. No rule will be fired unless the `run()` method is invoked on the Jess engine. To get the first and last names of the Authority from Jess in Java, the `fetch()` method is used. In our case, Jess uses the variables `AUTH-FIRST-NAME` and `AUTH-LAST-NAME` for this purpose.

```
 9 public class AuthorityServiceSoapBindingImpl
15         implements authority.AuthorityServicePortType {

  ...
30     public AuthorityResponse getNextAuthority(
31             AuthorityRequest req)
32         throws java.rmi.RemoteException {
33

  ...
44         String respCode = req.getResponseCode().toLowerCase();

  ...
52             getEngineInstance().store("RESPONSE-CODE",
53                     new Value(respCode, RU.SYMBOL));
54             getEngineInstance().batch("sws-jess-gen.v2.clp");
55             getEngineInstance().run();
56             Value authFirstName = getEngineInstance()
57                                         .fetch("AUTH-FIRST-NAME");
58             Value authLastName = getEngineInstance()
59                                         .fetch("AUTH-LAST-NAME");
```

List 7-6: Java code to invoke Jess

Our next objective is to allow Jess to communicate with Java. To do this, we also need to add Jess-specific codes to the Jess rules consolidated under the batch file "`sws-jess-gen.v2.clp`". In Jess, the `fetch()` function is used to retrieve the response code that comes from Java using the variable `RESPONSE-CODE` as shown in List 7-7. Conversely, the `store()` function in Jess is used to return the Authority's first and last name to Java as shown in List 7-7.

```
assert (response (fetch RESPONSE-CODE)))
```

List 7-7: Fetching code from Java in Jess

```
1 ; If the response is authrequired
2 ; then the next authority is Jackie-Brown.
3 (defrule Authority1
4     ?code <- (response authrequired)
5      =>
6         (printout t "It's Authority1" crlf)
7         (assert (authority Jackie Brown))
8         (store AUTH-FIRST-NAME Jackie)
9         (store AUTH-LAST-NAME Brown)
10        (retract ?code)
11        (facts)
12 )
```

List 7-8: Jess rule added with Jess-specific codes

We are now ready to deploy the Web service and the Jess rules in Apache Tomcat as described in Section 6.2.5.

For a complete list of all translated Jess rules, the Jess-specific codes added to the rules, the complete WSDL specification of the `AuthorityService.wsdl`, and the complete code of the Java SOAP Binding implementation, please see Appendix A.

7.3.5. Step 5 – Design of the Mediator PIM

Substep 1: Importing Jess rules as a Web service in ISDL

At this stage, we have transformed the business rules modeled in ARMOR into ACE at the CIM layer, ACE to RuleML at the PIM layer, and RuleML to Jess at the PSM layer. However, we still need to need to "lift" the platform specific information and behavior models from `AuthorityService.wsdl` similar to Step 1. Figure 7-16 shows the corresponding model in ISDL. We represent the `getNextOperation` as an operation execution as provides an appropriate Authority to the Mediator.

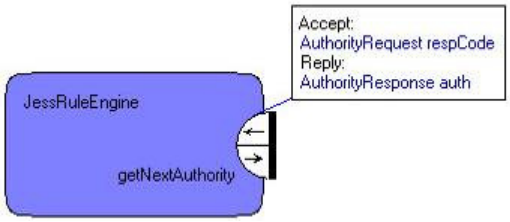


Figure 7-16: Abstracting `getNextAuthority` in ISDL

Consequently, Figure 7-17 shows the new `AuthorityService` in the business layer of ArchiMate realizing the business rule modeled in ARMOR. Now that all requirements can be satisfied by a service, we start designing the information and model behavior models of the Mediator in ISDL as will be described in the next step.

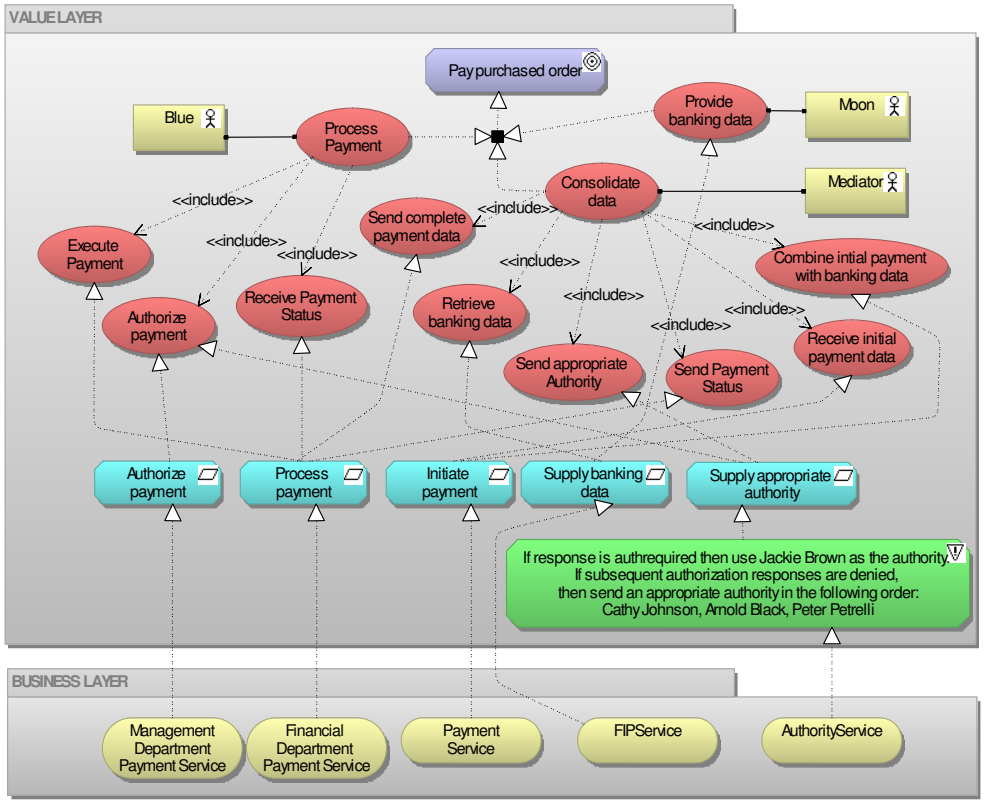


Figure 7-17: Realizing a business rule as a service in ArchiMate

Substep 2: Matching provided and requested services

This step essentially creates a skeletal design of the Mediator by matching provided and requested services between Blue, the Mediator and Moon. We do this by complementing the operations between them; that is, if a service is represented as an operation call in one behavior, this service will be represented as an operation execution in the other. For example, looking at Figure 7-5, the `initiatePayment` of Blue is represented as an *operation call* (requested service). Taking the complement of this operation on the Mediator's side, the `initiatePayment` will be represented as an *operation execution* (provided service).

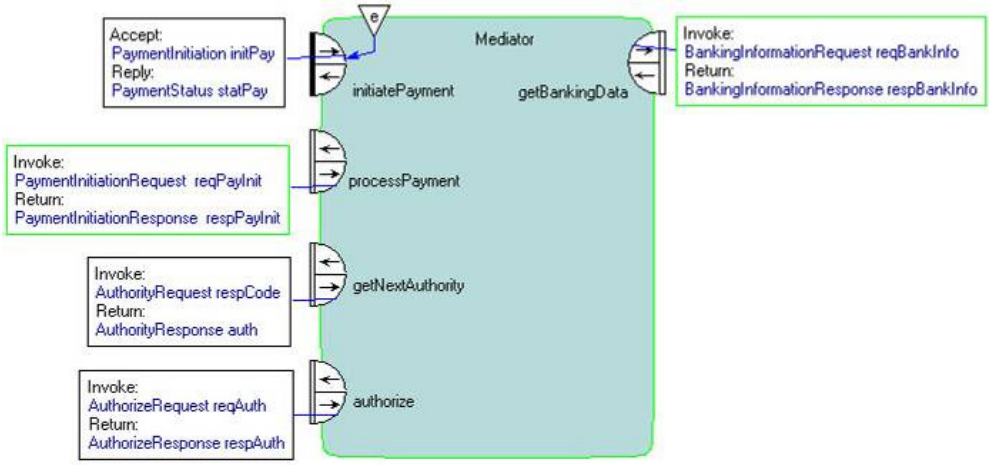


Figure 7-18: Matching provided and requested services on Mediator's side

Substep 3: Composing services by relating their operations

The objective of this step is to design the Mediator so that process mismatches between Blue and Moon are resolved. This can initially be done by matching the input information that is required by one operation to the output information that is produced by other operations. For example, one can see from the case description that a *causal relationship* can be established between Blue's `initiatePayment` and Moon's `getBankingData` because the latter requires a `requestId` which is provided by the former. As another example, a causal relationship can also be established between `getBankingData` and `processPayment`. The `getBankingData`'s response (which includes the `currencyCode`, `swiftCode`, and `accountNumber`) is required by Moon as part of the input message to invoke the `processPayment` operation. The causal relationship between operations is depicted in ISDL using an arrow as shown in Figure 66.

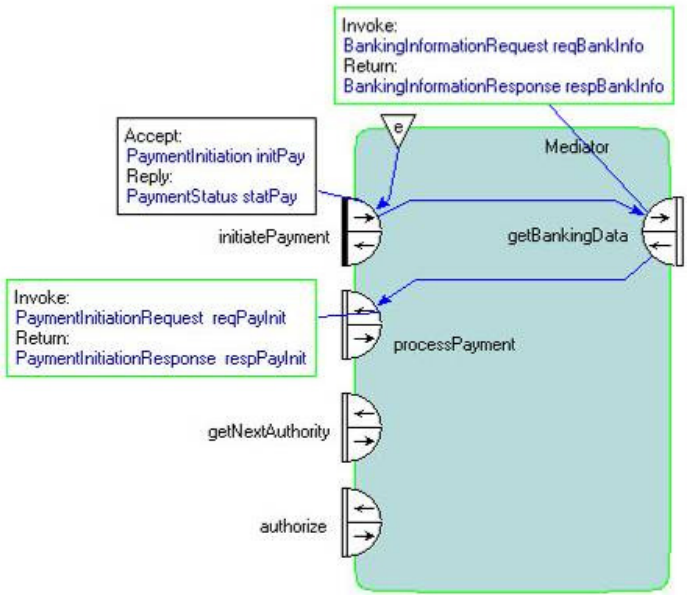


Figure 7-19: Composing services through operation parameter matching

Composing the services by relating the operations between provided and requested services may not be enough. Other specific processing logic may need to be modeled explicitly. To do this, the case description in Section 7.2 and the activity diagram shown in Figure 7-2 may provide additional information.

Figure 7-22 shows the complete behavior model of the Mediator and the mapping functions between operations. Figure 7-2 shows that the `processPayment` will have to be called twice: first when the purchase amount with less than `€2000`; and second when the purchase amount is greater than `€2000`. We model this separately in ISDL as `processPayment1stCall` and `processPayment2ndCall`, respectively. Although these operations are modeled separately, the same `processPayment` operation provided by Blue's Financial Department Payment Service is called.

Looking at the case description, the `authorize` and `getNextAuthority` operations must be invoked iteratively until the response from Management Department Payment Service's is `ACCEPTED`, it is best to move these operation calls into a separate behavior type so that it can be reused. This behavior type will thus need to be instantiated inside the Mediator. This also implies that the original operation calls will now be represented as *delegated operation calls* in the Mediator (depicted as a grayed operation call). The `Authorization` behavior type is in shown Figure 7-21 bearing the `authorize` and `getNextAuthority` operations.

Substep 4: Transforming data among the operation parameters.

From the matched requested and provided services in Substep 2, the message parameters of their operations will also be matched. This requires a definition of their data transformations describing how each input parameter is generated from the values of the output parameters. A Domain-Specific Language called *MDSL* has been developed for this purpose where a set of mapping functions perform data transformation between two or more objects.

Take for example the `payInit` mapping function that matches message elements between the `getBankingData` and `processPayment1stCall`. The semantic mapping between their data elements has been previously described in Step 2 (cf. Figure 7-7). Figure 7-20 shows the semantic mapping in MDSL.

```
22  mapping payInit{
23      target fd:PaymentInitiationRequest reqPayInit {
24          swiftCode       = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgt/finInstnId/bic";
25          currencyCode    = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgtAcct/ccy";
26          accountNumber   = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgtAcct/id/iban";
27          cd              = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgtAcct/tp/cd";
28      }
29      source moon:PaymentInitiation initPay{
30          reqPayInit      = "pain00100102";
31      }
32      source moon:BankingInformationResponse respBankInfo{
33          swiftCode       = "swiftCode";
34          currencyCode    = "currencyCode";
35          accountNumber   = "accountNumber";
36      }
37      expressions{
38          cd              = "CACC";
39      }
40  }
```

Figure 7-20: `payInit` mapping function in MDSL

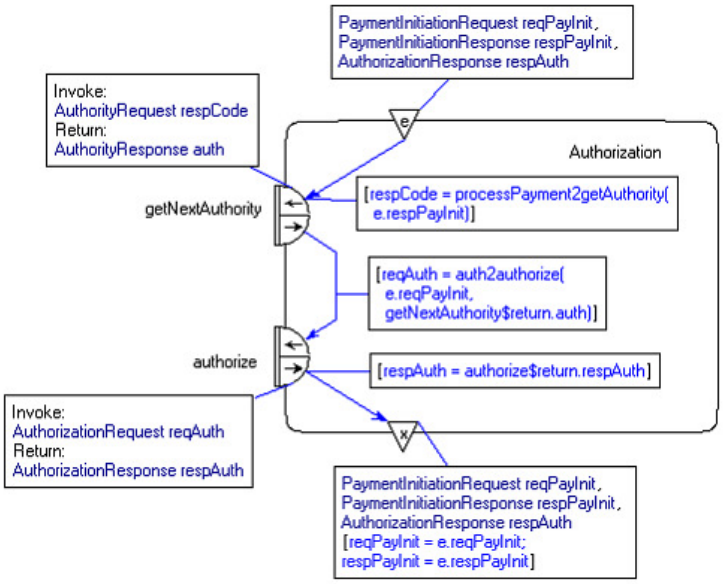


Figure 7-21: Authorization behavior type in ISDL

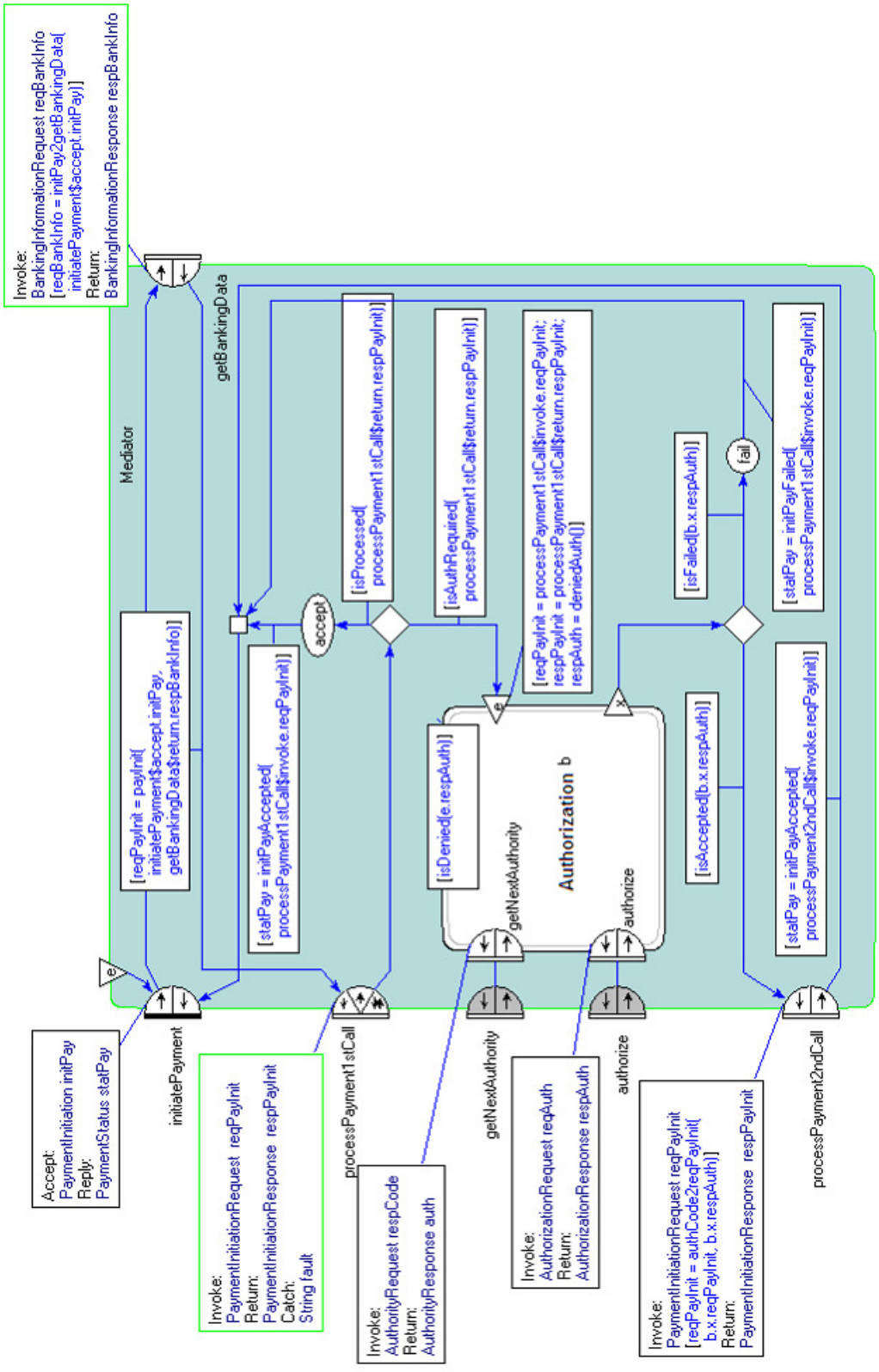


Figure 7-22: Behavior model of Mediator in ISDL

7.3.6. Step 6 – Validation of the Mediator PIM

For validation purposes, we simulate the behavior model using Sizzle. We validate two scenarios: firstly, when the purchase order amount is lower than `€2000`; secondly, when purchased order amount is above `€2000`. During simulation, Sizzle interacts with the SWS Challenge Test Bed provided by the SWS Challenge website[63].

Purchased order amount less than €2000

Figure 7-23 show the behavior model of the Mediator simulated in Sizzle. The Mediator begins its execution by accepting the purchased order (PO) amount via the `initiatePayment` operation which is invoked by Blue's Accounting Department System (ADS). The PO amount is part of the `PaymentInitiation` message shown here to be `€1067.54`. The `initiatePayment` thereafter causes the `getBankingData` operation to be executed (indicated by the arrow).

At this point, the Mediator acts as a service requester and invokes the `getBankingData` provided by Moon's Financial Information Provider (FIP) and passes a token as part of the `BankingInformationRequest` message. Moon replies to this request by returning `currencyCode`, `swiftCode`, and `accountNumber` as part of the `BankingInformationResponse` message. Here we show the `accountNumber` of `IE29AIBK93115212345678` returned by Moon (it is also possible to configure Sizzle to show the rest of the message elements).

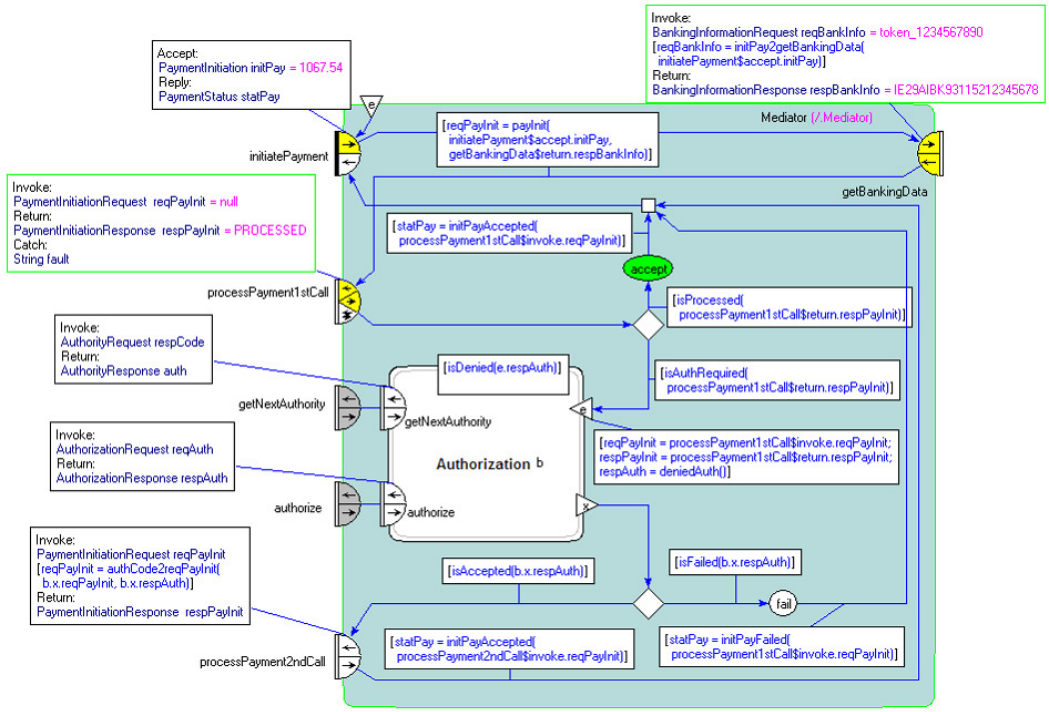


Figure 7-23: Mediator simulation in Sizzle: PO < €2000

Using the `payInit` mapping function, the Mediator thereafter combines the returned banking information with the `PaymentInitiationRequest` message to invoke the `processPayment1stCall` operation of Blue's ADS (described earlier in Figure 7-20). The `processPayment1stCall` operation returns a `PaymentInitiationResponseCode` of `PROCESSED` (part of the `PaymentInitiationResponse` message) since the PO amount is less than `€2000`. The mapping function `isProcessed` thus evaluates to true. The Mediator then creates the `PaymentStatus` message as its response to the `initiatePayment` invoked earlier. At

[63] http://sws-challenge.org/testbed/

this point, the SWS Challenge Test Bed receives the `PaymentStatus` message, and the Mediator terminates its execution.

Purchased order amount less than €2000

We now evaluate a scenario where the PO amount is greater than `€2000`. We use the PO amount of `€2590.54`. The Mediator proceeds with execution as with Figure 7-23. The difference with this scenario is that the `processPayment1stCall` operation returns a `PaymentInitiationResponseCode` value of `AUTHREQUIRED`. This is expected since the PO amount is greater than `€2000`. At this instance, the mapping function `isAuthRequired` thus evaluates to true requiring the Mediator to perform authorization request procedures.

Figure 7-24 shows the Mediator invoking the `getNextAuthority` operation of the Authorization behavior instance. Here the `responseCode` of `AUTHREQUIRED` will be initially used as part of the `AuthorityRequest` message. The Jess Engine begins sending an Authority starting from `Jackie Brown`. The Authorization behavior type then uses the returned authority to invoke the `authorize` operation of Blue's Management Department System (MDS).

Since the PO amount of `€2590.54` is well within the range that can be authorized by `Jackie Brown`, the `authorize` operation of MDS returns an appropriate `authorizationCode` and a `responseCode` of `ACCEPTED`. The execution thus exits the `Authorization` behavior instance, and causes the `isAccepted` mapping function to evaluate to true. This is shown in Figure 7-25.

The Mediator now attempts to invoke the `processPayment2ndCall` again, this time, with an appropriate `authorizationCode` as part of the `PaymentInitiationRequest` message shown in Figure 7-26. Blue's ADS should now return a `PaymentInitiationResponseCode` value of `PROCESSED`. This causes the `initPayAccepted` mapping function to assemble the `PaymentStatus` message with a `PaymentStatusCode` of `PI_ACCEPTED` which is thereafter returned to the SWS Challenge Test Bed. Finally, the Mediator terminates its execution.

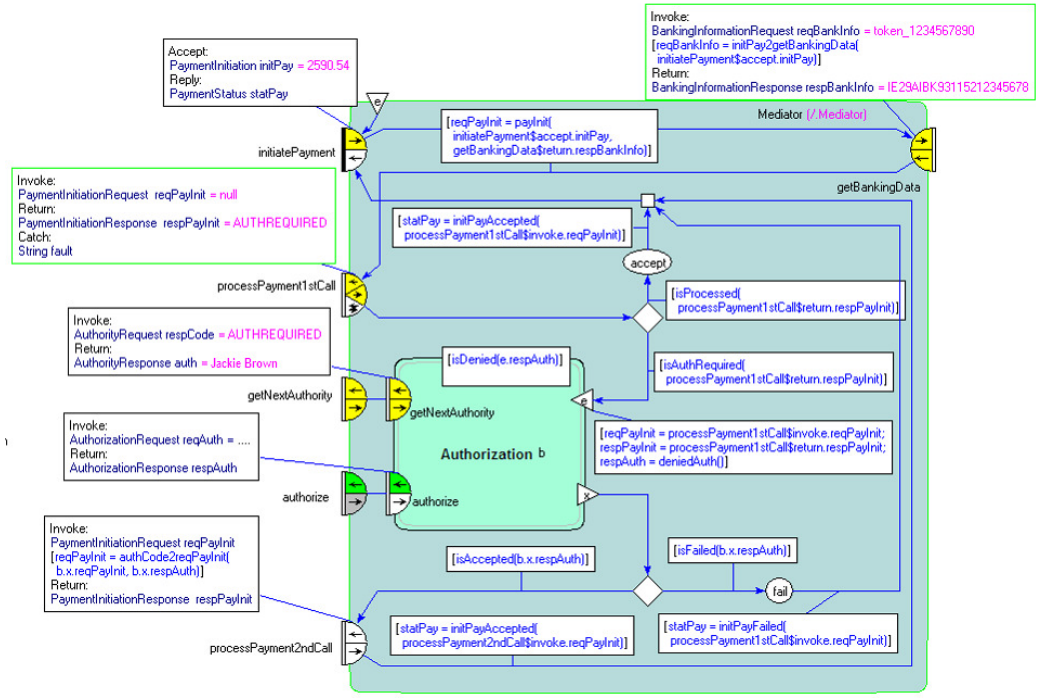


Figure 7-24: Mediator simulation in Sizzle: PO > €2000 (1)

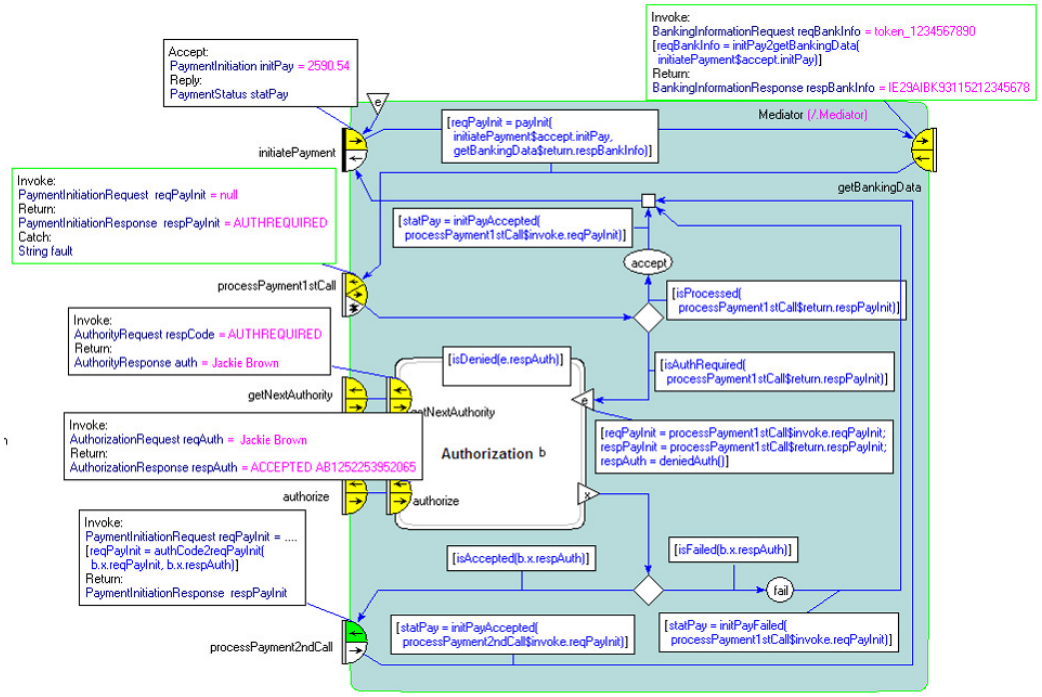


Figure 7-25: Mediator simulation in Sizzle: PO > €2000 (2)

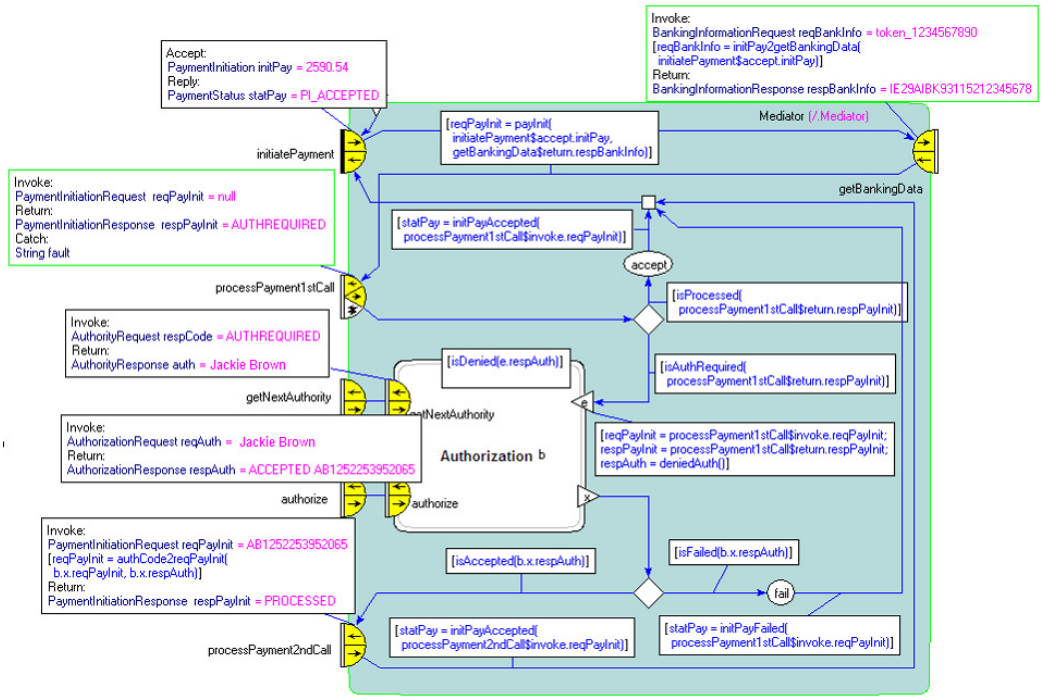


Figure 7-26: Mediator simulation in Sizzle: PO > €2000 (3)

7.4. The Enterprise Architecture

Figure 7-27 shows the overall Enterprise Architecture of the integration solution in ARMOR+ Archimate. The architecture paints a clearer, integrated, high-level view of how elements interact with one another at different layers to solve the main motivation of the integration which is to handle payment of purchased orders.

ARMOR + Archimate allow us continuity of modeling which is a feature that is absent from other goal modeling languages reviewed in this research (cf. Section 3.6). From the value layer where we model the motivations of the integration solution through goals, we are able to see how the goals are implemented in the business, application and technology layers – essentially allowing clearer requirements traceability analysis and business-IT alignment. For example, one can deduce from the architecture that to satisfy the integration goal of paying the purchased order, the `Supply appropriate authority` requirement must be satisfied. This requirement, in turn, can only be satisfied through the `Authority Service` available at the business layer. Conversely, one can also use the architecture to assess the impact should changes be introduced. For example, should another rule engine be used instead of Jess, one can see that the `Authority Service` will be affected, which in turn, affects the `Supply appropriate authority` requirement, and ultimately the primary goal of the integration.

At the business layer, we depict the equivalent ISDL behavior model of the Mediator as a business process in ArchiMate. Notice how the existing services are used by business activities of the Mediator business process (through the arrow relation construct). For example, the Authority Service is used by the `Get next higher Authority` business activity as part of the Mediator business process. We can also model how the business trigger `Send PO payment` is used by Blue to trigger the Mediator business process (shown through the arrow that connects them). This means that the Mediator business process will not execute unless Blue triggers its execution. Additionally, the constrain relation (depicted as an arrow with the `<<constrain>>` stereotype) tells us that the `Get next higher Authority` business activity of the Mediator business process is constrained by the business rule.

At the application layer, we show the interactions of application components and how they realize the services provided at the business layer (represented by the realization relation construct). The application layer also shows which component realizes the service provided at the business layer. For example, the Authority service is realized by the Jess Rule Engine application component.

7.5. Business rules as design artifacts

So far our approach to business rules has been to treat them as separate design artifacts: at the CIM layer we specify them in a near-natural controlled language, at the PIM layer we specified them in an XML-based rule specification, and at the PSM layer we represent them in an executable rule expression. To allow the business rules to be incorporated into and to constrain the design of the behavior model in ISDL, we have exposed them as an executable Web service. In Figure 7-22, the behavior model of the Mediator in ISDL accesses the business rule through the `getNextAuthority` operation.

Our solution has thus clearly separated the business rules from the business logic of the Mediator. Should there be new business rules that need to be created, the same process will need to be done to maintain the separation. Furthermore, should there be internal changes to the business rules (e.g. adding a new Authority), the behavior model of the Mediator should not be affected so long as the parameters of the operation remain the same. We believe that this feature adds to the flexibility and agility of our solution. As business rules are treated as separate design artifacts, we are able to treat them visibly and transparently; that is, they are not buried under

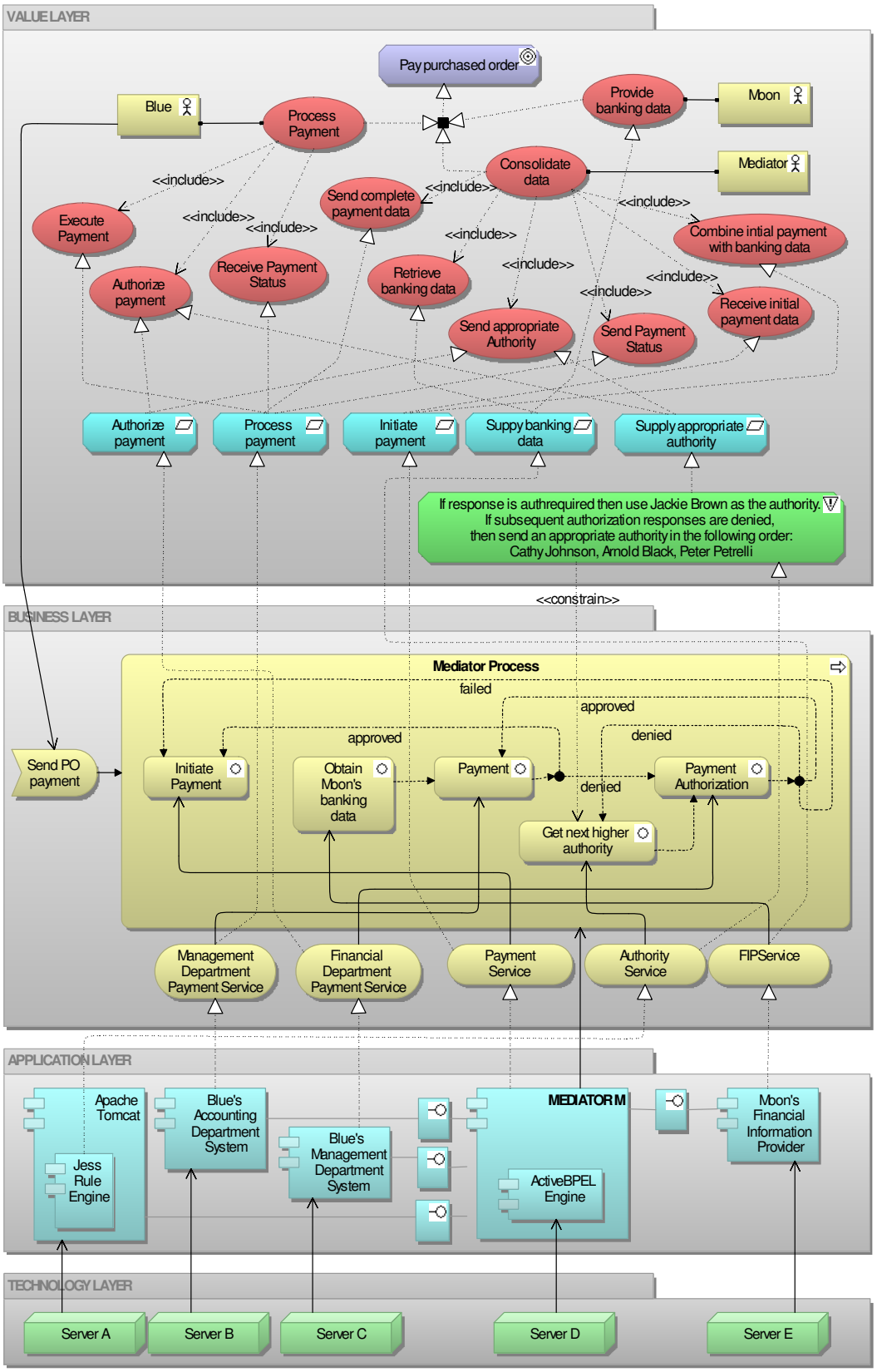


Figure 7-27: Mediator Enterprise Architecture in ARMOR+ArchiMate

some application code or under some business logic that is maintained by the Mediator – they can thus be managed better. Furthermore, as they are exposed as services, they are essentially reusable, and other business processes (or Mediators) may invoke them. Finally, as business rules can be stated in near-natural language, business domain experts are able to understand them better without having to look into the application code that implement them.

But how does the Mediator's behavior model in ISDL look like when the Authority-related rules are not treated as separate design artifacts? Figure 7-28 shows this. As there are no services that provide the ability to produce appropriate Authorities for a given purchase order, this requirement will have to be provided by the Mediator itself. And our previous approach to this has been to implement the rule using our Domain Specific Language. For example, the `createAuthList` mapping function creates the necessary code to initialize a list of authorities as shown in List 7-9. Other mapping functions will also need to be created to fetch the appropriate Authority.

```
mapping createAuthList{
    target util:List authList{
    }
    expressions{
        list        = createList();
        firstName1  = "Jackie";
        lastName1   = "Brown";
        firstName2  = "Cathy";
        lastName2   = "Johnson";
        firstName3  = "Arnold";
        lastName3   = "Black";
        firstName4  = "Peter";
        lastName4   = "Petrelli";
        authority1  = createAuth(firstName1,lastName1);
        authority2  = createAuth(firstName2,lastName2);
        authority3  = createAuth(firstName3,lastName3);
        authority4  = createAuth(firstName4,lastName4)
        authList    = addToList(list, authority1);
        authList    = addToList(list, authority2);
        authList    = addToList(list, authority3);
        authList    = addToList(list, authority4);
    }
}
```

List 7-9: Coding Authorities in MDSL

The business rule is therefore buried in the MDSL code, and hence not suitable for validation by business domain experts. Furthermore, the behavior model does not easily lend itself to being intuitively understandable as the business rule is not modeled explicitly. It is also unwieldy should there be changes to the business rules – the MDSL code will have to be rewritten, the necessary transformation from ISDL to BPEL recompiled, and the resulting BPEL specification redeployed.

7.6. Chapter summary

We have presented in this chapter a step-by-step account of our solution which is comprised of both a methodology and an architecture. We have used the SWS Challenge Payment Problem Scenario as a case study for illustration and validation purposes.

We have demonstrated that it is possible to incorporate goal modeling techniques to the drive the integration requirements in the mediation of services. And, through goal modeling languages such as ARMOR, we have demonstrated that business domain experts can take an active participation during the integration design.

Our methodology is largely a meet-in-the-middle approach where the identified requirements in ARMOR are matched with existing services. Requirements, which cannot be satisfied by any existing service but whose nature can be depicted as business rules, are exposed as Web services and then used to constrain the design the behavior of the Mediator. Business rules are first specified in ACE for easy validation among business domain experts, transformed into RuleML for interoperability, and finally into Jess for execution. Business rules are thus modeled explicitly, treated as distinct design artifacts, and separated from the business logic they constrain. We argue that such an approach to business rules design can lead to greater transparency, manageability, maintenance, isolation of changes, and thus agility of the solution.

Finally, we have shown that with ARMOR + Archimate, we are able to provide a clearer picture of the integration solution through its Enteprise Architecture. We are able to achieve continuity of modeling from requirements down to implementation, relate in an integrated manner how architectural elements contribute to the satisfaction of goals, and assess potential impacts should changes be introduced.

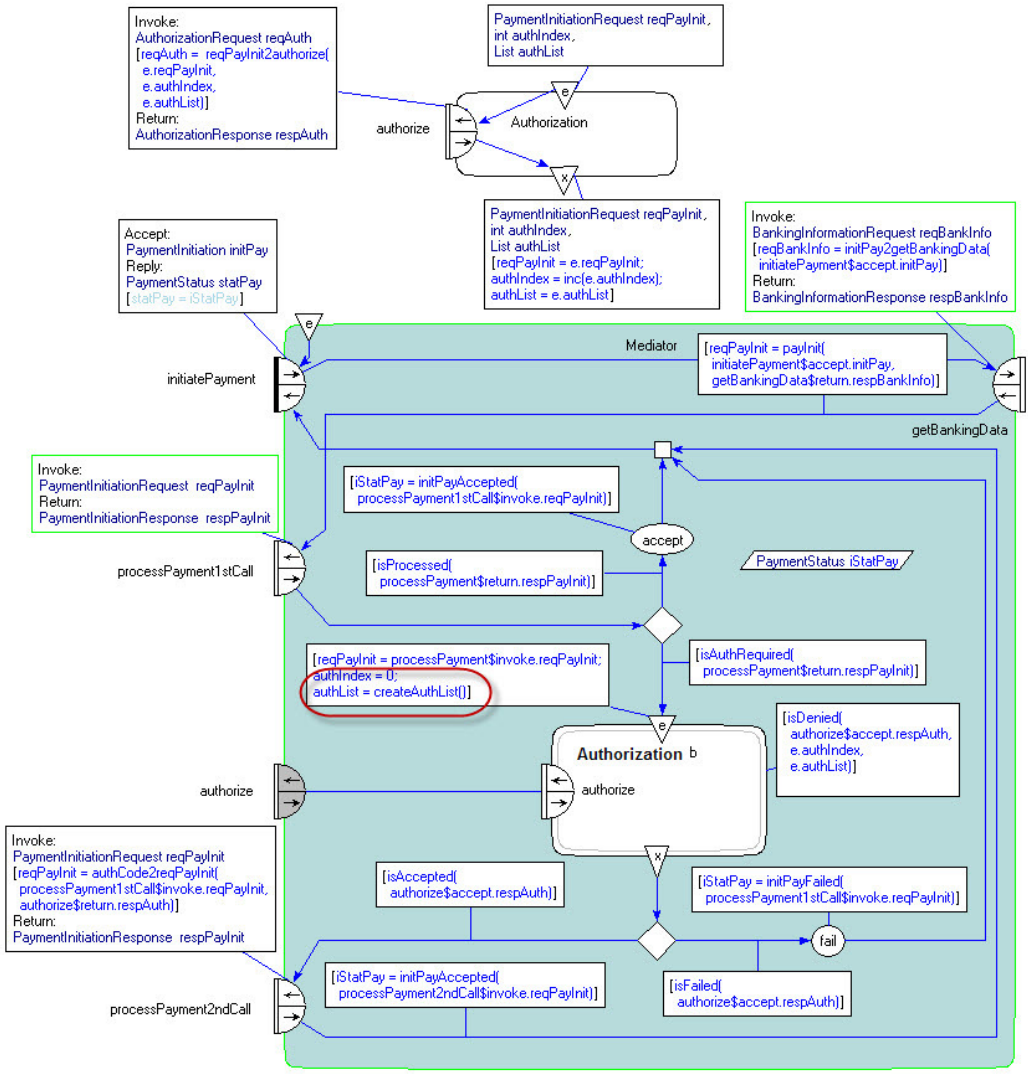


Figure 7-28: ISDL behavior model – before separating business rules

8

Final remarks

This chapter concludes this thesis by first providing a summary of answers to the research questions (Section 8.1), a list of contribution (Section 8.5), limitations of service mediation solution (Section 8.2), some possible directions for future work (Section 8.3), and recommendations to the Service Engineering Workbench of TNO (Section 8.4).

8.1. Conclusions

The revolving theme of this research has been to find a way to improve the participation of business domain experts in the design of networked enterprises. In particular, we approached the design of the networked of enterprises in the context of service mediation. Our hypothesis has been that through goal-driven and model-driven techniques, business domain experts can participate in the design of the mediation solution.

To answer this hypothesis, we have proposed a solution which is comprised of both a methodology and architecture based of the following three approaches:

- We involve business domain experts in the design of mediation solution by providing them concepts and tools to state their requirements through goal oriented requirements engineering using ARMOR.
- We adopt the COSMO framework for service mediation that uses model-driven, service-oriented and semantic web techniques.
- We use the framework proposed by Iacob, et al. (2009) that seeks to incorporate model-driven design principles as the binding agent in SOA development where high-level goals are refined and operationalized as business rules at various specifications at the MDA stack. These rules are then transformed into a language where they can be executed and eventually incorporated to constrain the design composition of services.

Our methodology takes a meet-in-the-middle approach to the design of service mediation. Existing service operations and messages exposed by the collaborating enterprises are first are identified and "lifted" from their platform specific representation (i.e. WSDL) into a platform independent model using ISDL (representing behavior models) and Java/UML (representing information models).

From this bottom-up approach, we then take a top-down approach where business domain experts are given the opportunity to identify, specify and structure their requirements using the ARMOR goal modeling language. From the resulting goal model, we take the requirements and match them with the existing services that can best satisfy them.

Requirements which cannot be satisfied by any existing service, but can be refined into business requirements are also modeled as such in ARMOR. These business rules are then treated as design artifacts so that they can be made executable and be used to constrain the design the Mediator. To do this, the business rules are first specified in a controlled language called ACE at the CIM layer. Business rules stated in near-natural language provides another opportunity for business experts to participate in the design process by validating the requirements of the integration as they are written in a form that is intuitively understandable. From ACE, business rules are transformed into their RuleML representations for added rule interoperability at the PIM layer. Finally, rules specified in RuleML are translated into Jess and wrapped in a Web service so that they can be executed at the PSM layer.

Together with the existing services and the business rules exposed as services, the Mediator can finally be designed in ISDL to handle process and data mismatches (PIM layer). After some validation activities such as behavior model simulation, the resulting ISDL model is then transformed into BPEL for execution and deployment (PSM layer). The validation of the Mediator PIM provides another opportunity for the business domain expert to participate in the integration design.

Our solution separates the business rules from the business logic of the Mediator they constrain. We believe this feature adds to the flexibility and agility of our solution. Should there be internal changes to the business rules, such changes are isolated from the behavior model of the Mediator so long as the parameters of the operation remain the same. Furthermore, when business rules are promoted as separate design artifacts, they can be made transparent (i.e. no longer hidden in application code or scattered elsewhere in the organization), reusable (i.e. other process can invoke them), and verifiable by business domain experts as they are stated in near-natural language.

We use ARMOR + Archimate as our framework for modeling Enterprise Architectures as it provides a high-level integrated view of all domains in the Enterprise Architecture. This allows us to take advantages of its two main benefits: requirements traceability and business-IT alignment. In particular, we are able to see how goals modeled in ARMOR are eventually realized by the underlying business, application and technological architectural components. Finally, we are able to see how the goals are implemented in the business, application and technology layers lending itself to better assessment of the impact should changes be introduced.

Summarizing the role of business domain experts in the design of the mediation solution, they can participate by:

- Identifying, structuring and specifying the integration requirements through goal-driven techniques in ARMOR.
- Provide semantic mapping to the message elements passed during the communication of collaborating services
- Specifying the business logic required to compose the collaborating service in the Mediator PIM.
- Validate the correctness of the translated business rule in ACE.
- Validate the correctness of the Mediator PIM in ISDL.

The following summarizes our responses to the research questions posted by this thesis:

RQ1: Who are the business domain experts? Why and in what ways should business domain experts be involved in the design of a service mediation solution?

We view business domain expert as the main stakeholder in the design of the mediation solution. They are people with an expertise in a certain business domain (e.g. healthcare, insurance, banking, etc.). They manifest their expertise by identifying and specifying the integration requirements. However, they do not necessarily have (or do not want to have) the technical knowledge to translate the business requirements into their technical implementations.

Their involvement is crucially important in the design of the mediation solution as several studies have shown that IT projects usually fail when users are not involved in specification of the requirements. Poor requirements essentially lead to poor software quality and costly repairs. Furthermore, their expertise is needed especially when the collaborating enterprise come from different domains (e.g. a healthcare system interoperating with an insurance system to process payments over healthcare claims.)

Business domain experts can participate in the design the mediation solution by serving as the source of the requirements, by validating the business requirements (which in our case can be done in two ways: by validating the resulting goal model), by validating the solution artifacts (which in our case include the translated business rules into ACE and the business logic of the Mediator and), by providing semantic information to the mapping of data elements exchanged by the operations of collaborating systems, and by specifying modifications to the requirements.

RQ2: What are goals? How are they used to specify software requirements? What goal-driven approaches are there? How can goal-driven approaches be used to specify requirements for the design of the mediation solution?

Goals are prescriptive statements that describe the motivations of the mediation solution. They provide a way to specify in a disciplined and structured manner the objectives (or the "whys") of the system at various levels in the enterprise. The cooperation of some agents (e.g. organizational actors, operators, end users, etc.) is needed to satisfy the achievement of the goal(s).

We use Goal Oriented Requirements Engineering (GORE) activities such as Goal identification/analysis, goal decomposition/refinement and goal identification techniques to elicit, elaborate, structure, specify, analyze, negotiate, document, and modify the integration requirements in a structured manner. We also use such techniques to refine business goals into business rules.

Among the surveyed GORE approaches (i.e. i*, KAOS, BMM, and ARMOR) we find ARMOR to be the most suitable to specify the integration requirements because of its tight connection with the Archimate enterprise modeling framework. ARMOR + Archimate allow us to take advantages of the benefits of better business-IT alignment and requirements traceability. In particular, ARMOR provides a way to describe how goal models are supported by architectural services that realize them at the various layers of the enterprise. Furthermore, the integrated modeling of architectural domains allows better understanding from both business domain experts and IT experts. Finally, we are able to achieve continuity of modeling from requirements layer down to their business, technical, and finally physical implementations which is absent from other goal modeling approaches.

RQ3: What is service mediation? What service mediation approaches are currently available? How do they involve business domain experts in the design of an mediation solution?

Service mediation involves the design of a software component that sits in between two or more interoperating systems faced with the problem of incompatible process and data specifications. Process mismatch occurs when systems use services that define different ordering of message exchanges. Data interoperability mismatch occurs when systems use different information models (or vocabularies) to describe the messages that are exchanged by their services.

Surveying current service mediation approaches (i.e. DERI, SWE-ET, jABC/jETI, and COSMO approaches), we find two salient observations: Firstly, although such approaches follow model-driven techniques, the technologies and methodologies they use are limited only to the PIM level. They use technologies and techniques that are still too technically-oriented, and thus lean more closely to the technology side. This essentially means that business domain experts, whom we consider as those without sufficient technical background, still need to be familiar with the technologies that are used to design the Mediator. Thus, there is still a need to bring the design of mediation solutions closer to business domain experts. Secondly, although most of the approaches support the importance of requirements specification, they do not describe how these requirements are identified and refined in a structured manner so as to guide the composition of the Mediator service.

RQ4: How can goal-driven and service mediation approaches be combined to allow business-level design of the service mediation solution? How can model-driven techniques be used to combine these approaches?

We involve business domain experts in the design of mediation solution by providing them concepts and tools to state their requirements using ARMOR. We have chosen ARMOR largely because of its suitability for the design of enterprise architectures, ability to document, communicate and reasons about requirements, and provisions for an easy-to-use set of language constructs. Its integration with the Archimate modeling framework for enterprise architecture also allows us to perform better business-IT alignment and requirements traceability analyses.

We extend the COSMO framework for service mediation by introducing goal-driven approaches. Using ARMOR, we specify how the composition will be achieved by first modeling the goals at the CIM level through goal analysis/identification, decomposition/refinement, or abstraction techniques. We then use ArchiMate to model how these goals are implemented and realized by the underlying Enterprise Architecture.

Once we have the requirements of the service mediation solution expressed in terms of goals driven by business requirements, we use the framework proposed by Iacob, et al. (2009) for goal-driven design of service-oriented systems using model-driven techniques. The framework seeks to incorporate a business view in SOA development where high-level goals are refined and operationalized as business rules. These rules are then transformed into a language where it can be executed and eventually incorporated into the design and composition of services. MDA is the central binding concept that allows goals to be transformed into business rules, to services, and finally to executable rule expressions.

RQ5: What available tools are there to help in the (semi-)automated Mediator design, goal modeling, and model transformations?

Under the Design Space, we use BiZZDesign Architect to model architectural components at the business, application, and technology layers in ArchiMate. We use Grizzle to design the behavior model of the mediator in ISDL. The simulation of the

behavior ISDL model can be done using Sizzle. We use Tizzle to map the semantic data mismatches message elements. Tizzle can also be used to transform the XSD schemata to JAXB for execution inside the Eclipse environment. The generated JAXB files can then be reverse engineered to arrive at their UML specifications using Omondo EclipseUML.

Under the Goal and Business Rule Space, we use ARMOR to model goals and relate them to the architectural components using BiZZDesign Architect. The APE Web client gives us an editing environment to test the correctness of our ACE sentences before we translate them into RuleML transformation using ACE2RRML. We created a prototype to transform the generated RuleML specifications into Jess. We used the Oxygen XML editor to code the XSLT stylesheet transformer. Finally, we have used Eclipse WTP to create the necessary Java bean skeletons to wrap the Jess rules and deploy them as a Web service in Apache Tomcat.

RQ6: What requirements can be drawn from the service mediation solution which can serve as inputs to the design of the Service Engineering Workbench of TNO-ICT?

We provide an answer to this question in Section 8.5.

8.2. Contributions

The following lists the contributions made by this research:

- Proposed a solution, comprising of a methodology and an architecture, which allows business domain experts to drive the design of their integration requirements. By using ARMOR to model the motivations of the integration solution through goals, business domain experts can specify their requirements better ensuring greater success in the implementation of the requirements. Aside from identifying requirements, our research has demonstrated that business domain experts can help in providing semantic mapping to the message elements passed during the message exchanges, specify the business logic required to compose the Mediator PIM, validate the correctness of the translated business rule in ACE, and validate the correctness of the Mediator PIM in ISDL.
- Showed that it is possible to combine goal-driven approaches to drive the design of the service mediation solution by using model-driven development techniques as the binding paradigm.
- Proposed an extension to the original COSMO methodology for service mediation by introducing ARMOR as a goal modeling language at the CIM layer to engineer requirements and drive the service mediation solution.
- Demonstrated the use of ARMOR in combination of ArchiMate as a suitable framework for modeling integration requirements as goals and translating them into architectural artifacts at the different layers of the Enterprise Architecture.
- Showed that ARMOR and ArchiMate can provide sufficient concepts and tooling support to design the requirements of the of the mediation solution modeled at the business level, and subsequently translating them into their architectural and technical implementation. This we believe facilitates better alignment with business and IT and requirements traceability.
- Showed concretely, through the framework of Iacob, et al., (2009), that business rules can be combined with the design of service-oriented systems. With the framework, we have demonstrated that promoting business rules as first class citizens in the requirements world can lead to a more agile, transparent, and manageable solution.

- Showed that it is possible to specify business rules at the CIM layer using ACE, RuleML at the PIM layer, and Jess and PSM layer. We have also shown that transformations between specifications are possible.
- Contributed to the SEW project by providing some initial design recommendation based on the output of this thesis, and possibly some other opportunities with other projects at TNO.
- Contributed to the SWS Challenge Workshop by providing a solution to its Payment Problem Scenario.
- Created an Eclipse Plugin editor called Tizzle to lessen the problem of error-prone and time-consuming coding of matching data specifications using the domain specific language used by COSMO.
- Created a prototype for transforming rules expressed in RuleML into Jess using XSLT.
- Addressed the gaps in the available tools between business rules and service-oriented design tools at the CIM and PIM levels including their transformations (depicted by the question mark at the upper right-hand corner of Figure 8-1) as described in the position paper of Iacob and Jonkers (2008). Our contribution is highlighted by the red rectangle with rounded corners in the same figure.

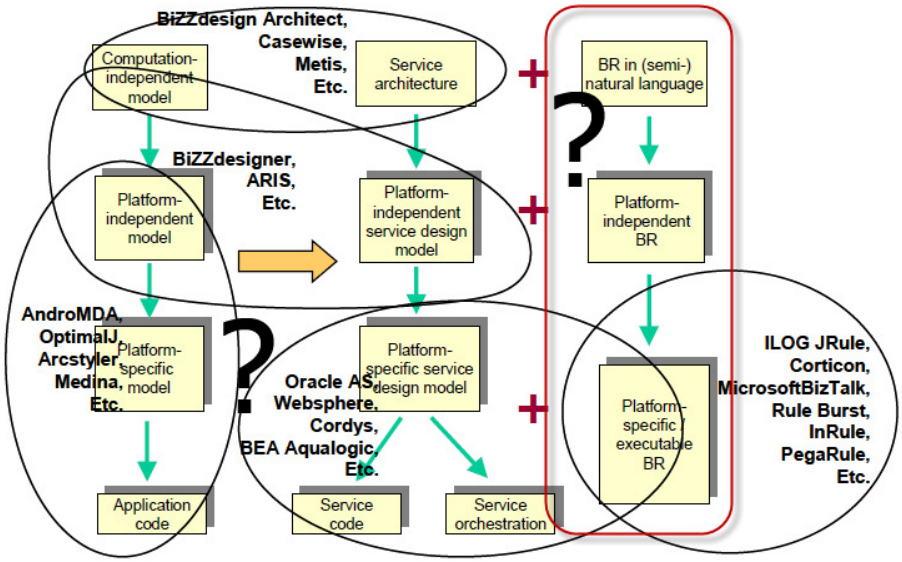


Figure 8-1: Tool coverage: gaps and issues (Iacob and Jonkers, 2008)

8.3. Limitations

The following are the limitations evident in this research:

- Our prototype for transforming RuleML to Jess is rather case-specific. Although we have strived to create just one XSLT stylesheet to transform all the four RuleML Authority rules, we have not explored the possibility of applying our transformation to other possible rule structures. While doing this research, our experience has been that once a different DRS representation is generated by the APE parser for a given rule stated in ACE, necessary (and in fact time-consuming) changes need to be done to the XSLT stylesheet as well.
- Goal modeling is not only a scientific but a creative process as well. It could very well be that some business domain experts may have different manners of structuring the goal models.

The methodology we proposed at in Step 2 (Modeling goals in ARMOR) is still at its early stages and may require further refinement once ARMOR has been applied to more case studies both in the industry and the academe.

- Our research has not explored validation between the generated goal models at the CIM layer and their implementations in the PIM and PSM layers. Similarly, in the Goals and Business Rules Space, the semantic equivalence that should be preserved between different specifications of the rules at the different layers of the MDA stack depends solely on the quality and correctness of the transformations we have used.
- We have used a multitude of tools for our solution. One evident limitation to our research has been the lack of an integrated environment where we are able to perform all the necessary design and transformations in one (or less) development environment(s).
- Corollary to the previous limitation, our solution also uses a multitude of technologies each with their own unique (or sometimes overlapping) concepts. This may prove to be a challenge for stakeholders, especially the IT expert, as the learning curve may be quite high.

8.4. Future work

The following lists some possible improvements to our solution:

- As our transformation from RuleML to Jess is largely a prototype, possible research work can be directed towards this end. One of the possible solutions is the open source project initiated by Benjamin Grosof of MIT Sloan, et al., back in 2001 called the *Semantic WEb Enabling Technology (SWEET) Rules*[64] which provides an integrated set of translation tools for semantic Web rules and ontologies between such technologies as RuleML, Jess, XSB, CommonRules, Jena, OWL, and SWRL, via XML and RDF using Java. A component to the SWEET Rules project is *SweetJess* that provides transformation from RuleML to Jess and back. Installing the product is however very difficult and quite discouraging because of the number of dependencies required to run the tool. The source code of the product is also not available. Continued support and improvement to the tool has been discontinued (as per our correspondence with the project initiator).
- Since business rules have now been treated as separate design and implementation artifacts, one potential problem could the management challenges that can result when the number of business rules grows. This gave rise to the concept of a *rule registry* which is similar to how Web services are management by *Universal Description Discovery and Integration* (UDDI)[65] technologies. A rule registry (or rule repository) should provide, among others, the ability to list all rules that have been defined and their current status, identify the relation of these rules with other rules especially if they form a set, identifies the original author and those who made changes to them, provide information as to their location (Morgan, 2002). Apart from the work of Morgan (2002), there are already some research work directed towards this end like the one of Giurca, Diaconescu, Pascalau and Wagner (2008) where they created a basic architecture of a Web-based registry rules that allows rule sets discovery (through their metadata properties such as URI and last modified date).
- We have only validated our mediation solution via a fictional case study derived from the SWS Challenge scenarios. A possible future research would be to validate the solution from an industrial environment where real stakeholders (i.e. business domain experts, business analysts, and IT expert) participate in the design of integration requirements.

[64] http://sweetrules.projects.semwebcentral.org
[65] http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

- ARMOR does not currently provide a standard "way of working" in modeling goals. More mature goal modeling approaches, like KAOS for example, propose a set of heuristic steps which include among others some best practice development techniques, and an extensive collection of formally-driven goal refinement patterns (see Van Lamsweerde, 2009, p. 309). This could be a possible improvement for ARMOR.

- As ARMOR supports the concept business rules which can either be represented as design artifacts or requirements, ARMOR should be improved to allow translation of modeled business rules into controlled language such as ACE or SBVR. There are some current efforts to translate goal modeling language into a rule-based specification; for example, the work of Milanović, Kaviani, Gašević, Giurca, Wagner, Devedžić, and Hatala (2007) attempts to translate KAOS, not to a controlled language though, but to RuleML.

- As our solution shows, goals and the business rules modeled at the CIM layer will have to be used as basis for the design of the behavior model at the PIM level. This research, however, does not investigate their validation.

  In terms of the derived business rules, for example, we have not investigated in a formal way whether or not the semantic equivalence between the different specifications of the business rules at the different layer of the MDA stack are preserved. Although the validation from ACE to DRS translation has been well investigated, the validation of the translation from DRS to RuleML and Jess still needs some formal verification.

  In terms of the services realizing the requirements modeled in ARMOR, we still need formal ways to verify if whether or not the overall effect of the service provided by the Mediator, when executed, does indeed satisfy the overall achievement of the integration's hard goal.

  These could be potential topics for further research work. Figure 8-2 shows the validation graphically in relation to the overall methodology depicted in Figure 5-14.

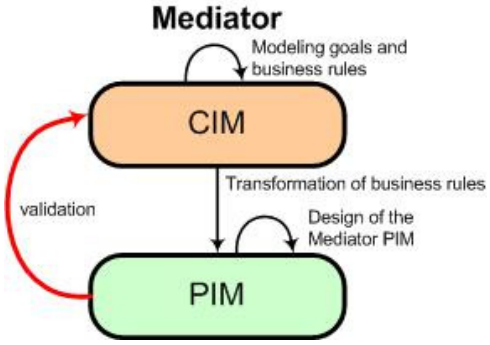


Figure 8-2: Validating goals and business rules

- As an alternative to the rule technologies we have used in this thesis, a possible future work could be to investigate the use Semantics of Business Vocabulary and Business Rules or SBVR (Object Management Group, 2008a) as the controlled language to specify rules at the CIM layer. Some key differences between ACE and SBVR include:

  - Unlike ACE, SBVR is more expressive: business rules can be specified not only in English but in other languages as well through what is known as a *speech community*.

  - SBVR also has the concept of *semantic community* where a set of *business vocabulary* is used by members of the community as basis for a common understanding of the things that they have to deal with (e.g. people in the airline industry all agree to the definition of "passenger"). ACE words are domain independent; that is, the meaning represented by the words is up to the user of ACE – words to ACE are just syntactical elements.

- SBVR also support some formalisms that are absent in ACE; for example, modalities (e.g. "*it is necessary that*…", "*it is possible that*…") which may add flexibility to stating business rules (not just by mere `IF…THEN` statements)

However, as SBVR is quite a recent specification adopted by the OMG (adopted January 2008), there is currently very little research done to translate SBVR to XML-based rule technologies. So far, we have only found one work describing SBVR transformation to R2ML (Demuth and Liebau, 2007). The study is however at its preliminary stages and no mature tool support is provided.

Furthermore, tool support for SBVR coding is limited: one for example is SBEAVER[66] – an open source Eclipse Plugin Editor for SBVR; this project, however, is not actively maintained (last build was in 2006).

Finally, our solution is largely based on MDA (which in itself is a standard of OMG). SBVR could be a viable alternative to ACE as SBVR is aligned with MDA principles.

At the PIM layer, there are currently active proposals available: one is the *Rule Interchange Format (RIF)*[67] spearheaded by the World Wide Web Consortium (W3C)[68] which seeks to recommend a standard for rules interchange in rule-based systems on the semantic web; another is the one of the Object Management Group (OMG)[69] called the *Production Rule Representation (PRR)*[70] which is a standard that proposes a common vendor-independent representations of production rule in UML. It is not currently clear however whether these standards will merge in the future as there are similarities between the two. Also, there seems to be very few research work done in the transforming SBVR into these technologies.

8.5. Recommendations to the SEW

Overview of SEW

The Service Engineering Workbench (SEW) (Hofman, 2008b) aims to provide a tooling environment that allows technology-independent, business-viewpoint design and choreography of services in organizational networks. Information (semantic) and behavior (process) requirements of business transactions, interactions, and protocols between service consumers and providers serve as inputs to the SEW. These requirements are modeled at the business level which can later be transformed into technology specifications and implementations which serve as its outputs.

The SEW defines a business service as that which is offered by a business role (or actor) to many other actors. A business service is used by one or more business transactions initiated by an actor. A business transaction thus involves a particular sequence of business interactions between two roles: business service consumer and business service provider. The business interaction sequence must adhere to a predefined business service protocol. A business service protocol is applicable to one or more business services. A business interaction is an instance of a business interaction type. Figure 8-3 shows the conceptual model of the SEW.

Recommendations

The SEW strives to separate the design from the technology. This is consistent with the nature of our solution. As we have mentioned, we have strived to keep our model-driven methodology

[66] http://sbeaver.sourceforge.net/
[67] http://www.w3.org/2005/rules/wiki/RIF_Working_Group
[68] http://www.w3.org/
[69] http://www.omg.org
[70] http://www.omg.org/spec/PRR/1.0/Beta1/index.htm

problem-oriented and technology independent; that is, several other technologies may be used in lieu of those we have selected here. For example, SBVR may be used as the controlled language of choice instead of ACE. The UML and Java representations of the information models may be replaced with more advanced ontology matching algorithms which can also be supported by the SEW. With all of these technology choices, our methodology essentially remains the same.

As we have mentioned, one of the limitations of this research is that we have used a multitude of tools which are largely isolated from each other (cf. Chapter 6). We have, however, already identified the functionalities that we require from these tools. The workbench can thus serve as an integrated development environment that provides all of the functionalities we require for the design and specification goals, the design of the Mediator, and business rules at the different layers of the MDA, including their transformations.

Grizzle does not currently support the modeling of information models using ontologies, it would a nice functionality for SEW to provide an environment where both the design of the information and behavior models of the mediator can be done in one place.

Our methodology (including the available tools; e.g. Grizzle) currently supports only the *manual* service composition design of the Mediator to resolve process and data mismatches. One improvement to the SEW would be the ability to derive the Mediator behavior and information models automatically which can be quite a challenging task.

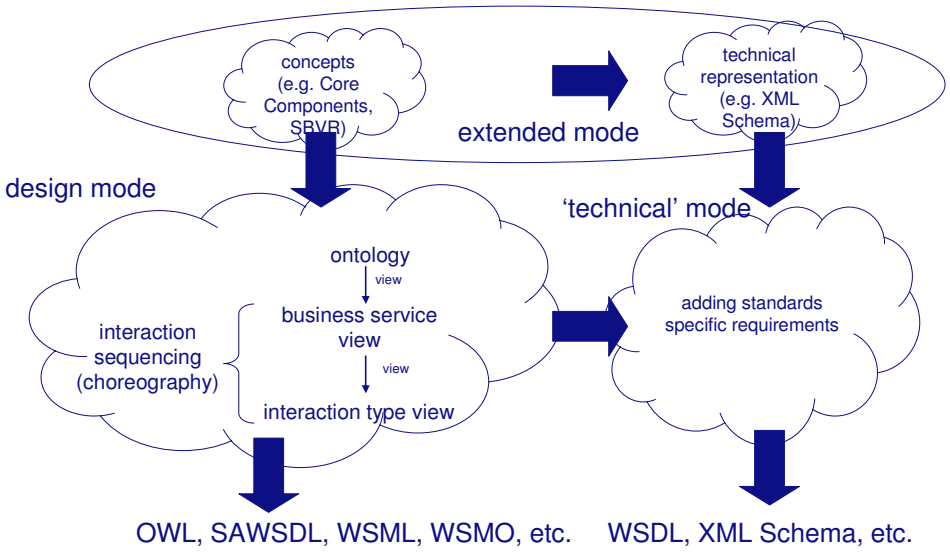


Figure 8-3: Conceptual model of the SEW (Hofman, 2008b)

References

Alexander, I.F. (2007). Choosing a Tram Route: An Experience in Trading-Off Constraints. *Proceedings 15th IEEE International Requirements Engineering Conference, New Delhi,* October 15-19, 2007, 350-355.

Alge, B.J., & Upright, K. (2009). Needs analysis for HRIS. In M. Thite & M. J. Kavanagh (Eds), HRIS: Basics, Applications And Directions. Thousand Oaks, Sage.

Alexander, I., & Robertson, S. (2004). Understanding Project Sociology By Modeling Stakeholders, *IEEE Software, IEEE Computer Society Press*, 21(1), 23–27.

An, L., & Jeng, J-J. (2007). Business-Driven SOA Solution Development. In: *IEEE International Conference on e-Business Engineering (ICEBE 2007),* 439-444.

Anton, A.I. (1996).Goal-based requirements analysis, In: *Second IEEE International Conference on Requirements Engineering (ICRE `96)*, Colorado Springs, Colorado, 136-144.

Anton, A.I., McCracken, W.M., & Potts, C. (1994). Goal decomposition and scenario analysis in business process reengineering, In: *Proceedings of the 6th International Conference on Advanced Information Systems Engineering*, 94-104.

Bahr, P. (2008). The ACE2RRML Web Service – A Web Service for Translating Controlled Natural Language into Reaction RuleML. Retrieved August 4, 2008 from http://www.pa-ba.info/?q=pub/ace2rrml.

Bass, L., Clements, P., & Kazman, R. (2003). Software Architecure in Practice. *Addison-Wesley*. ISBN: 0-321-15495-9.

Bell, T.E., & Thayer, T.A. (1976). Software Requirements: Are They Really a Problem? *Proceedings of the ICSE-2: 2nd International Conference on Software Engineering*, San Francisco, 61-68.

Brambilla, M., Celino, I., Ceri, S., Cerizza, D., Della Valle, E., & Facca, F. M. (2006). A Software Engineering Approach to Design and Development of Semantic Web Service Applications, In: *Proceedings of the 5th International Semantic Web Conference (ISWC-2006)*, Athens, GA, USA, 5-9 November 2006, LNCS 4273, 172-186.

Brambilla, M., Stefano, C., Della Valle, E., Facca, F.M., & Tziviskou, C. (2008). A Software Engineering Approach Based on WebML and BPMN to the Mediation Scenario of the SWS Challenge. In: *Semantic Web Services Challenge, Semantic Web And Beyond*, 8, 51. ISBN 978-0-387-72495-9. Springer US.

Braye, L., Ramel, S., Grégoire, B., Leidner, S., & Schmitt, M. (2006). State of the Art Business Rules Languages, EfficientSwift/D3.1, *Public Research Centre Henri Tudor*, Luxembourg.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, & J.(2004). Tropos: An Agent- Oriented Software Development Methodology, *Autonomous Agents and Multi-Agent Systems*, 8(3), 203-236.

Business Process Modeling Notation (BPMN) - FAQ. (2004). Retrieved June 17, 2009, from http://www. bpmn.org/Documents/FAQ.htm

Business Rules Group (2000). Defining business rules – what are they really? Business Rules Group. Retrieved July 7, 2009, from http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf.

Ceri, S., Gottlob, G., & Tanca, L. (1989). What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions of Knowledge and Data Engineering*, 1(1), 146-166.

Ceri, S., Fraternali, P., & Matera, M. (2002). Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing*, 6(4).

Charfi, A., & Mezini. M. (2004). Hybrid web service composition: Business processes meet business rules. In: *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC)*, pp. 30–38.

Dardenne, A., Van Lamsweerde, A., & Fickas, S. (1993). Goal-Directed Requirements Acquisition, *Science of Computer Programming*, 20, 3-50.

Darimont, R. & Van Lamsweerde, A. (1996). Formal Refinement Patterns for Goal-Driven Requirements Elaboration, In: *Proc. FSE'4 -Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering,* San Francisco, 179-190.

Deng, X. (2006). Intentional Modeling for Enterprise Architecture - Managing Knowledge about "Why" to Support Change. Master's Thesis, *Faculty of Information Studies, University of Toronto.*

Demuth, B., Liebau, H. (2007). An Approach for Bridging the Gap Between Business Rules and the Semantic Web. In: *Proc. Int. Symposium on Advances in Rule Interchange and Applications (RuleML'07)*, LNCS 4824 (2007) 119-133.

Dirgahayu, T. (2005). *Model-Driven Engineering of Web Service Compositions: A Transformation from ISDL to BPEL*. Master Thesis. University of Twente, Enschede, The Netherlands, July 2005.

Dirgahayu, T., Quartel, D.A.C., & van Sinderen, M.J. (2007). Development of Transformations from Business Process Models to Implementations by Reuse. In: *Proceedings of MDEIS 2007 - 3rd International Workshop on Model-Driven Enterprise Information Systems, pp. 41-50, in conjunction with the 9th International Conference on Enterprise Information Systems (ICEIS 2007)*. Funchal, Portugal, 12 June 2007.

van Eijck, J. (2005). Discourse representation theory. In K. Brown (Ed.), *Encyclopedia of Language and Linguistics*. Amsterdam: Elsevier.

Eindhoven, van T.E. (2008). Increasing Flexibility by Combining Business Processes with Business Rules, MSc Thesis, University of Twente, in cooperation with TNO-ICT, Enschede, The Netherlands.

Erl, T. (2005). Service-Oriented Architecture: Concepts, Technology, and Design. *Prentice Hall PTR*. ISBN: 0-13-185858-0.

Engelsman, W. (2008). A Method for Requirements Management and Modeling. Master Thesis, Business Information Technology, *University of Twente*, Enschede, The Netherlands.

Friedman-Hill, E. (2003). Jess in Action: Rule-Based Systems in Java, *Manning Publications Co.*

Friedman-Hill, E. (2008). Jess The Rule Engine for the Java Platform Manual for version 7.1p2. Sandia National Laboratories. Retrieved July 12, 2009 from http://www.jessrules.com /jess/docs/Jess71p2.pdf and http://www.jessrules.com/jess/docs/71/.

Fuchs, N., Schwertel, U., & Schwitter, R. (1999). Attempto Controlled English (ACE) Language Manual Version 3.0. Institut für Informatik der Universität Zürich.

Fuchs, N.E., Hofler, S., Kaljurand, K., Schneider, G., & Schwertel, U. (2005). Extended Discourse Representation Structures in Attempto Controlled English. *Technical Report i-2005.08, Department of Informatics, University of Zurich*, Zurich, Switzerland, 2005.

Fuchs, N. E., Kaljurand, K., & Schneider, G. (2006). Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. *In FLAIRS 2006*, Melbourne Beach, Florida, May 11-13, 2006.

Fuchs, N.E., & Kaljurand, K. (2006). Attempto Controlled English: Language, Tools and Applications Logical Background [PowerPoint slides]. *University of Zurich*, Retrieved September 21, 2009, from http://attempto.ifi.uzh.ch/site/courses/files/ACE.Course. UniZH.1206.Logical_Background.pdf

Fuchs, N., Kaljurand, K., & Kuhn, T. (2009). Discourse Representation Structures for ACE 6.5. Technical Report ifi-2009.04. University of Zurich. Retrieved August 13, 2009 from http://attempto.ifi.uzh.ch/site/pubs/papers/drs_report_65.pdf

Giurca, A., Diaconescu, I.-M., Pascalau, E., & Wagner, G. (2008). On the Foundations of Web-Based Registries for Business Rules, In: *Intelligent Distributed Computing, Systems and Applications*, Springer Berlin / Heidelberg, 162/2008, 251-255.

Halle, von B. (2002). Business Rules Applied: Business Better Systems Using the Business Rules Approach. *John Wiley & Sons*, New York.

Hasselwanter, T., Kotinurmi, P., Moran, M., Vitvar, T., Zaremba, M. (2006). WSMX: a Semantic Service Oriented Middleware for B2B Integration, In *Proceedings of the 4th International Conference on Service Oriented Computing* , Springer-Verlag LNCS series, December 2006, Chicago, USA

Hirtle, David. (2006). Translator: A Translator from Language to Rules. Retrieved August 4, 2009, from http://www.ruleml.org/translator.

Hofman, W.J. (2008a). *Service Engineering Workbench – Overall and First Graduation Project*. Unpublished Manuscript. TNO, The Netherlands.

Hofman, W.J. (2008b). *Requirements and Standards Support* [PowerPoint slides]. Unpublished manuscript, TNO, The Netherlands.

Iacob, M.-E., Jonkers, H. (2008). A Model-Driven Perspective on the Rule-Based Specification of Services. In: *12th International IEEE Enterprise Distributed Object Computing Conference, 2008. EDOC '08*, pp.75-84, 15-19 Sept. 2008

Iacob, M.-E., Rothengatter, D., & van Hillegesberg, J. (2009). A Health-care Application of Goal-driven Software Design. *Applied Medical Informatics*, 24(1).

Institute of Electrical and Electronics Engineers (IEEE). (1990). IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. *IEEE*. New York, USA.

Jackson, M. (1995). Software Requirements & Specifications – A Lexicon Of Practice. Principles And Prejudices, *ACM Press*, Addison-Wesley.

Jureta, I. J., Faulkner, S., & Schobbens P. Y.(2006). A More Expressive Softgoal Conceptualization for Quality Requirements Analysis. In*: Conceptual Modeling - ER, 25th International Conference on Conceptual Modeling,* Arizona, USA.

Kaljurand, K., Fuchs, N.E. (2006). Bidirectional mapping between OWL DL and Attempto Controlled English. *In Fourth Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro.

Kamp, H., & Reyle, U. (1990). From Discourse to Logic: An Introduction to the Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, *Kluwer*, Dordrecht.

Kardasis, P., & Loucopoulos, P. (2005). A Roadmap for the Elicitation of Business Rules in Information Systems Projects. *Business Process Management Journal*, 11(4), 316–348.

Keller, S.E., Kahn, L.G., & R.B. Panara. (1990). Specifying Software Quality Requirements with Metrics, In: R.H. Thayer, & M. Dorfman, (Eds.), *Tutorial: System and Software Requirements Enginering*, IEEE Computer Society Press, 145-163.

Kueng, P., and Kawalek, P. (1997). Goal-Based Business Process Models: Creation And Evaluation. *Business Process Management Journal*, 3(1), 17-38.

Kubczak, C., Margaria, T., Steffen, B., & Nagel, R. (2008). Service-oriented mediation with jABC/jETI. In: C. Petrie, T. Margaria, M. Zaremba, H. Lausen. (Eds). *Semantic Web Services Challenge—Results From The First Year*. Springer, Berlin, 71–99.

Lapouchnian, A. (2005). Goal-oriented Requirements Engineering: An Overview of the Current Research, *Depth Report*, University of Toronto.

Lankhorst, M., et al. (2005). Enterprise Architecture at Work. *Springer-Verlag Berlin Heidelberg*. ISBN-10 3-540-24371-2.

Lamsweerde, A. van, R. Darimont, & Massonet, P. (1995). Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt, In: *Proc. RE'95 - 2nd Intl. IEEE Symposium on Requirements Engineering*, March 1995, 194-203.

Lamsweerde, A. van. (2000). Requirements Engineering in the Year 00: A Research Perspective. Invited Keynote Paper, *Proc. ICSE'2000: 22nd International Conference on Software Engineering*, ACM Press, 5-19.

Lamsweerde, A. van. (2001). Goal-Oriented Requirements Engineering: A Guided Tour. In: *Proceedings 5th International Symposium on Requirements Engineering (RE'01)*, 1-13. IEEE.

Lamsweerde, van A. (2003). From System Goals to Software Architecture, In *Formal Methods for Software Architectures, M. Bernardo & P. Inverardi* (eds.), LNCS 2804, Springer-Verlag.

Lamsweerde, A. van. (2008). Requirements Engineering: From Craft To Discipline. In: SIGSOFT '08/FSE-16: Proceedings Of The 16th ACM SIGSOFT International Symposium On Foundations Of Software Engineering, 238-249, New York, NY, USA, 2008. ACM.

Lamsweerde, A. van. (2009). Requirements Engineering – From System Goals to UML Models to Software Specification. John & Wiley and Sons, England.

Letier, E., & Lamsweerde, van A. (2002). Deriving Operational Software Specifications from System Goals, In: *Proc. FSE'10: 10th ACM SIGSOFT Symp. on the Foundations of Software Engineering*, Charleston

Letier, E., Kramer, J., Magee, J., & Uchitel S. (2006). Deriving Event-Based Transition Systems from Goal-Oriented Requirements Models. Technical Report 02/2006, *Imperial College London*.

Lister, T. (2009). World-class Business Analyst? Retrieved August 30, 2009 from http://www. volere. co.uk/worldclass.htm.

Miller, J. & Mukerji, J. (Eds.). (2003). MDA Guide Version 1.0.1, *Object Management Group* doc.omg/2003-06-01, 12 June 2003, Retrieved June 12, 2009 from http://www.omg.org/docs/omg/03-06-01.pdf.

Margaria, T., Winkler, C., Kubczak, C., Steffen, B., Brambilla, M., Cerizza, D., Ceri, S., Della Valle, E., Facca, F., & Tziviskou, C. (2007). The SWS Mediator With Webml/Webratio And Jabc/Jeti: A Comparison. In *Proc. ICEIS'07, 9th Int. Conf. on Enterprise Information Systems*, Funchal (P), June 2007.

Mantovaneli Pessoa, R., Quartel, D.A.C, & van Sinderen, M. (2008). A Comparison of Data and Process Mediation Approaches. In: *Second International Workshop on Enterprise Systems and Technology (I-WEST-2008)*, May 23, 2008, Enschede, The Netherlands.

Mantovaneli Pessoa, R., van Sinderen, M.J., & Quartel, D.A.C. (2009). Towards Requirements Elicitation In Service-Oriented Business Networks Using Value And Goal Modelling. In: *Proceedings of the 4th International Conference on Software and Data Technologies (ICSOFT 2009)*, 26-29 July 2009, Sofia, Bulgaria. 392-399. INSTICC. ISBN 978-989-674-009-2

Milanović, M., Kaviani, N., Gašević, D., Giurca, A., Wagner, G., Devedžić1, V., & Hatala, M. (2007). Business Process Integration by using General Rule Markup Language. In: *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, p. 353-364.

Morgan, T. (2002). Business Rules and Information Systems: Aligning IT with Business Goals. *Addison Wesley*.

Morris, J. (October 20, 2005). The Zen of Jess II, Retrieved July 19, 2009, from http://www.Jessrules.com/Jess/zen.shtml.

Mylopoulos, J., Chung, L., & Yu, E. (1999). From Object-Oriented to Goal-Oriented Requirements Analysis, *Communications of the ACM*, 42(1), 31-37.

Muschamp, P. (2004). An introduction to Web Services, *BT Technology Journal*, 22(1), 9-18 (January 2004).

Nicolae, O., & Wagner, G. (2008). Verbalizing R2ML rules into SBVR. In: *Proc. of 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2008).* IEEE. September, 26-29, 2008. pp. 181–189. Timisoara, Romania

Nagarajan, M., Verma, K., Sheth, A.P., & Miller, J.A. (2007). Ontology Driven Data Mediation in Web Services. *International Journal of Web Services Research*, 4(4).

Narayanan, S., McIlraith, S. (2002). Simulation, Verification and Automated Composition of Web Services. *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii.

Object Management Group. (2008a). *Semantics of Business Vocabulary and Business Rules (SBVR), v1.0*, OMG Document Number formal/08-01-02. Retrieved June 17, 2009 from http://www.omg.org/spec/ SBVR/1.0/PDF.

Object Management Group. (2008b). *Business Motivation Model Specification version 1.0*. Retrieved August 25, 2009, from http://www.omg.org/spec/BMM/1.0/PDF.

Papazoglou, M, & Ribbers, M.A. (2006). E-Business: Organizational and Technical Foundations. *John Wiley & Sons, Ltd.* England.

Papazoglou, M. (2007). What's in a Service? In: F. Oquendo (Ed.), *Software Architecture, First European Conference, ECSA 2007, Aranjuez, Spain, September 24-26, 2007* (LNCS vol. 4758, pp. 11-28), Springer-Verlag Berlin Heidelberg, ISBN 978-3-540-75131-1

Pokraev, S., Quartel, D.A.C., Steen, M.W.A., Wombacher, A., & Reichert, M. (2007). Business Level Service-Oriented Enterprise Application Integration. In *Proceedings of the 3rd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2007),Funchal, Portugal, March 28-30, 2007* (pp 507-518). Berlin: Springer Verlag.

Pokraev, S., & Reichert, M. (2006). Mediation Patterns for Message Exchange Protocols. *In: Proc. Of CAiSE'06 Workshops/Open INTEROP Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP)*, Luxembourg, Presses Universitaires de Namur, 659-663.

Pokraev, S., Reichert, M., Steen, M.W.A., Wieringa, R.J. (2005). Semantic and Pragmatic Interoperability: A Model for Understanding. In: *J. Castro, E. Teniente (Eds.) Proceedings of the CAiSE'05 Workshops*, Porto, Portugal, June 2005, pp. 377-382.

Quartel, D.A.C. (2009a). A Requirements Modelling Language for Enterprise Architectures: ARMOR for Requirements Modelling. Novay, Technical Report TI/RS/2009/007, Enschede, The Netherlands.

Quartel, D.A.C, Dirgahayu, T., & M. van Sinderen. (2009b). Model-Driven Design, Simulation and Implementation of Service Compositions in COSMO. In: *International Journal of Business Process Integration and Management*, Vol. 4, No. 1.

Quartel, D.A.C., Engelsman, W., Jonkers, H., & van Sinderen, M. (2009c). A Goal-oriented Requirements Modelling Language for Enterprise Architecture. In: *Proceedings of 13th IEEE International EDOC Conference (EDOC 2009)*, Auckland, New Zealand, 31 August - 4 September 2009 (To appear).

Quartel, D.A.C, Pokraev, S., Mantovaneli Pessoa, R., & van Sinderen, M. (2008a). Model-Driven Development of a Mediation Service. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, 117–126, Washington, DC, USA. IEEE Computer Society.

Quartel, D.A.C., Pokraev, S., Dirgahayu, T., Mantovaneli Pessoa, R., & van Sinderen, M.J. (2008b). Model-driven, Semantic Service Integration using the COSMO approach. (2008).In *Proceedings of 7th International Semantic Web Services Challenge Workshop, in conjunction with the 7th International Semantic Web Conference (ISWC 2008)*, Karlsruhe, Germany.

Quartel D.A.C, Steen, M.W.A., Pokraev, S., & van Sinderen, M.J. (2007a). COSMO: A Conceptual Framework For Service Modelling and Refinement. Information System Frontiers (on-line), March 2007. Extended Version Of: A Conceptual Framework For Service Modelling. *Proc. of the 10th IEEE Enterprise Distributed Object Computing (EDOC) Conference*, Hong Kong, China, 319-330.

Quartel, D.A.C. & van Sinderen, M. (2007b). On Interoperability and Conformance Assessment in Service Composition. In: *Proceedings of the Eleventh IEEE International EDOC Enterprise Computing Conference (EDOC 2007)*, pp. 229-240.

Quartel, D.A.C., Dijkman, R., & van Sinderen, M. (2004). Methodological Support for Service-Oriented Design with ISDL. In *Proceedings of 2nd International Conference on Service Oriented Computing*, New York, NY, USA

Quartel, D.A.C., Ferreira Pires, L., & van Sinderen. M. (2002). On Architectural Support for Behavior Refinement in Distributed Systems Design. *Journal of Integrated Design and Process Science,* 6(1).

Respect-IT. (2007). A KAOS Tutorial. Retrieved August 26, 2009, from http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf.

Ribarić, M., Gašević, D., Milanović, M., Giurca, A., Lukichev, S., & Wagner, G. (2008). Model-Driven Engineering of Rules for Web Services . In R. Lämmel, J. Vissers, and J. Saraiva (Eds.), *Generative and Transformational Techniques in Software Engineering II*, LNCS 5235, 377-395.

Robertson, S. & Robertson, J. (2006). Mastering the Requiremements Process. *Addison Wesley Professional*. ISBN-10: 0-321-41949-9.

Robinson, S. (1997). Simulation Model Verification and Validation: Increasing the User's Confidence. *Proceedings of the 29th Conference on Winter Simulation*, 53-59.

Rolland, C., & Salinesi, C. (2005). Modeling Goals and Reasoning with them. In Y. Aurum & C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 189-217). Springer Berlin Heidelberg.

Roman, D., et al. (2005). Web Service Modeling Ontology. *Applied Ontologies*, 1(1), 77-106.

Rosenberg, F., Nagl, C., & Dustdar, D. (2006). Applying Distributed Business Rules – The VIDRE Approach, In: *Proceedings of the IEEE International Conference on Services Computing (SCC'06)*.

Rosenberg, F., Dustdar, S. (2005). Business rule integration in BPEL - a service-oriented approach. In: *Proceedings of the 7th International IEEE Conference on E-Commerce Technology*.

Rudolph, G. (July 10, 2008). Some Guidelines for Deciding Whether to Use a Rules Engine, Retrieved July 19, 2009, from http://www.Jessrules.com/Jess/guidelines.shtml.

Ryan, K., & Greenspan, S. (1996). Requirements Engineering Group Report, *Succeedings of IWSSD8 - 8th Intl. Workshop on Software Specification and Design, ACM Software Engineering Notes*, September 1996, 22-25.

Sadiq, S., Orlowska, M., Sadiq, W., & Foulger, C. (2004). Data Flow and Validation in Workflow Modeling. In *Proceedings of the 15th Australasian Database Conference*, 207–214.

Shiffman, R.N., Michel, G., Krauthammer, M., Fuchs, N.E., Kaljurand, K., & Kuhn, T. Controlled Natural Language for Clinical Practice Guidelines. *Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009), CEUR Workshop Proceedings*, 2009.

Sinderen, M. van, (2008). Challenges and Solutions in Enterprise Computing (Editorial). *Enterprise Information Systems*, 2(4), 341-346.

Soffer P., & Wand Y. (2005). On the Notion of Soft Goals in Business Process Modeling, *Business Process Management Journal*, (11)6, 663-679

Steffen, B., Margaria, T., Nagel, R., J¨orges, S., & Kubczak, C. (2006). Model-Driven Development with the jABC. In *HVC - IBM Haifa Verification Conference*, LNCS N.4383. Springer Verlag.

Schwitter, R. (4th February 2009). Controlled Natural Languages. Retrieved August 12, 2009 from http://sites.google.com/site/controllednaturallanguage/

Spek, J. van der, & Steen, M.W.A. (2001). Electronic Procurement of Temporary Labour: Guidelines for the Implementation of E-Procurement Procedures for Temporary Labour. *Telematica Instituut*, TI/RS/2001/129, Enschede, The Netherlands.

Standish Group. (2001). Extreme Chaos. Standish Group International, http://www.standish group.com/sample_research/PDFpages/q3-spotlight.pdf.

Taveter, T., & Wagner, G. (2001). Agent-Oriented Enterprise Modeling Based on Business Rules. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER'01)*, 527-540.

Vissers, C. A., Ferreira Pires, L., Quartel, D. A. C., and van Sinderen, M. J. (2007). The Architectural Design of Distributed Systems. Reader for the Design of Telematics Systems. *University of Twente*.

Vongsavanh, A., & Campbell, C. (2008). The Roles and Skill Sets of Systems vs Business Analysts. *Proceedings of the 19th Australasian Conference on Information Systems*, December 3-5 2008, Christchurch, New Zealand.

Wagner, G. (2002). How to design a general rule markup language? In: *XML Technologien f¨ur das Semantic Web – XSW 2002, Proceedings zum Workshop*, 24-25 Juni 2002, Berlin.

Wagner, G., Antoniou, G., Tabet, S., & Boley, H. (2004). The Abstract Syntax of RuleML - Towards a General Web Rule Language Framework. In: *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence(WI 2004)*. IEEE Computer Society. 20-24 September, 2004. pp. 628–631. Beijing, China

Wagner, G., Giurca, A., & Lukichev, S. (2005). A General Markup Framework for Integrity and Derivation Rules, In: Principles and Practices of Semantic Web Reasoning, F. Bry, F. Fages, M.

Marchiori, & H.J. Ohlbach, In Dagstuhl Seminar Proceedings, Dagstuhl, Germany, (2005). Forschungszentrum f¨ur Informatik (IBFI), Schloss Dagstuhl, Germany

Wagner, E., & Piccoli, G. (2007). Moving Beyond User Participation to Achieve Successful IS Design. *Communications of the ACM*, 50(12), 51-55.

Wang, J.W., Han, Y.B., Wang, J., & Li, G. (2004). An Approach to Dynamically Reconfiguring Service-Oriented Applications from a Business Perspective. In: *Proceedings of Advanced workshop on content computing*. LNCS 330, 15–17. Springer, Heidelberg.

Wang, J., Yu, J., & Han, Y. (2005). A Service Modelling Approach with Business-Level Reusability and Extensibility, In *Proc. IEEE Int'l Workshop on Service-Oriented System Engineering (SOSE2005)*, Beijing, China 2005.

Wiegers, K.E. (2006). More About Software Requirements: Thorny Issues and Practical Advice. *Microsoft Press*. ISBN: 0735622671.

Wieringa, R. (2004). Requirements Engineering: Problem Analysis And Solution Specification (Extended Abstract). *Lecture Notes in Computer Science*, 3140, 13-16.

Wieringa, R. (2007). *Writing a Report about Design Research*, University of Twente, Enschede. Retrieved June 9, 2009, from http://wwwhome.cs.utwente.nl/~roelw/ReportingAbout DesignResearch.pdf

Wieringa, R. (2008). Research and Design Methodology for Software and Information Engineers. *Department of Electrical Engineering, Mathematics, and Computer Science, University of Twente*, Enschede, The Netherlands.

White, S. A. (2004). Business Process Modeling Notation (BPMN), *BPMI.org*, http://www.bpmi. org/bpmi-downloads/BPMN-V1.0.pdf

Ying, L., & Li, W. (2007). An Intelligent Service Composer for Business-level Service Composition. In: *The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC-EEE 2007)*, 567-570, 23-26 July 2007.

Yu, E. S. K. & Mylopoulos, J. (1998). Why Goal-Oriented Requirements Engineering? In: E. Dubois, A. L. Opdahl & K. Pohl (Eds), *Proceedings of the 4th International Workshop on Requirements Engineering: Foundation for Software Quality (RESFQ 1998)*, Presses Universitaires de Namur, 15-22.

Yu, E., Strohmaier, M., & Deng, X (2006). Exploring Intentional Modeling and Analysis for Enterprise Architecture. In *Proceedings of the EDOC 2006 Workshop on Trends in Enterprise Architecture Research (TEAR 2006)*

Yu, E. (1997). *Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering*. Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97). Jan. 6-8, 1997, Washington D.C., USA, pp. 226-235.

Yu, E. (1995). Modelling Strategic Relationships for Process Reengineering. PhD thesis, *University of Toronto, Department of Computer Science*.

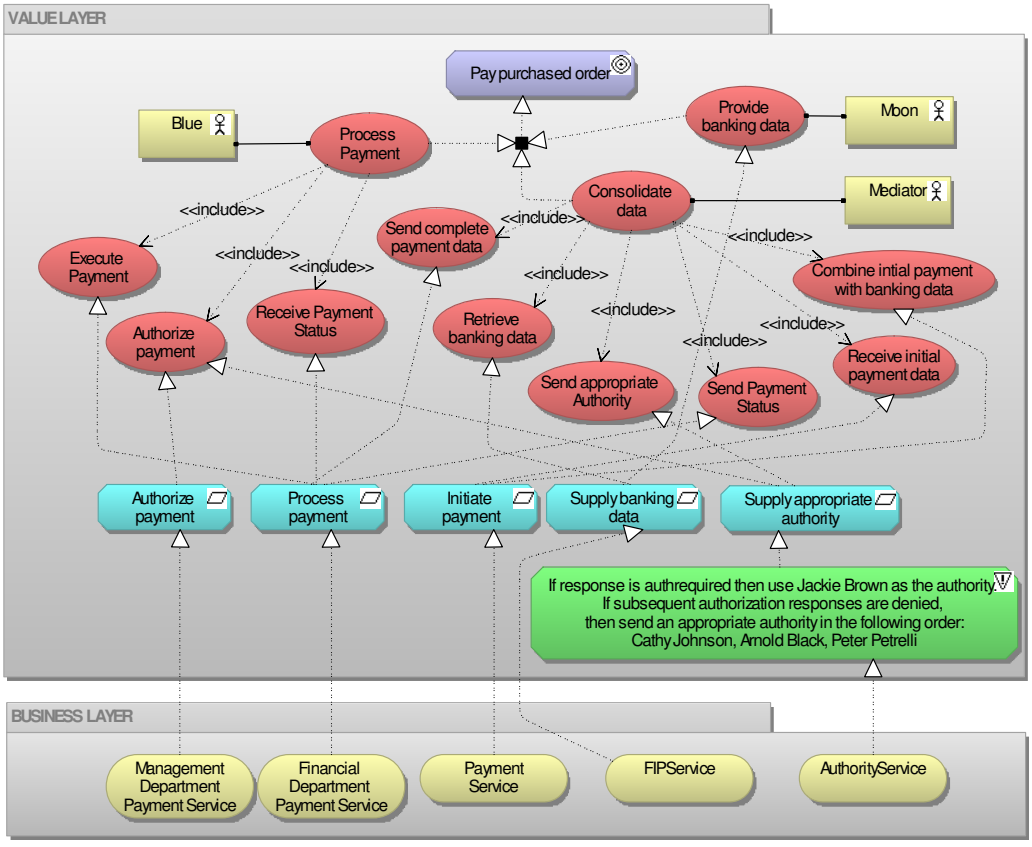
Zaremba, M., Herold, M., Zaharia, R., & Vitvar, T. (2008). Data and Process Mediation Support for B2B Integration. In: *Evaluation of Ontology-based Tools and the Semantic Web Service Challenge Workshop at ESWC 2008*.

Appendices

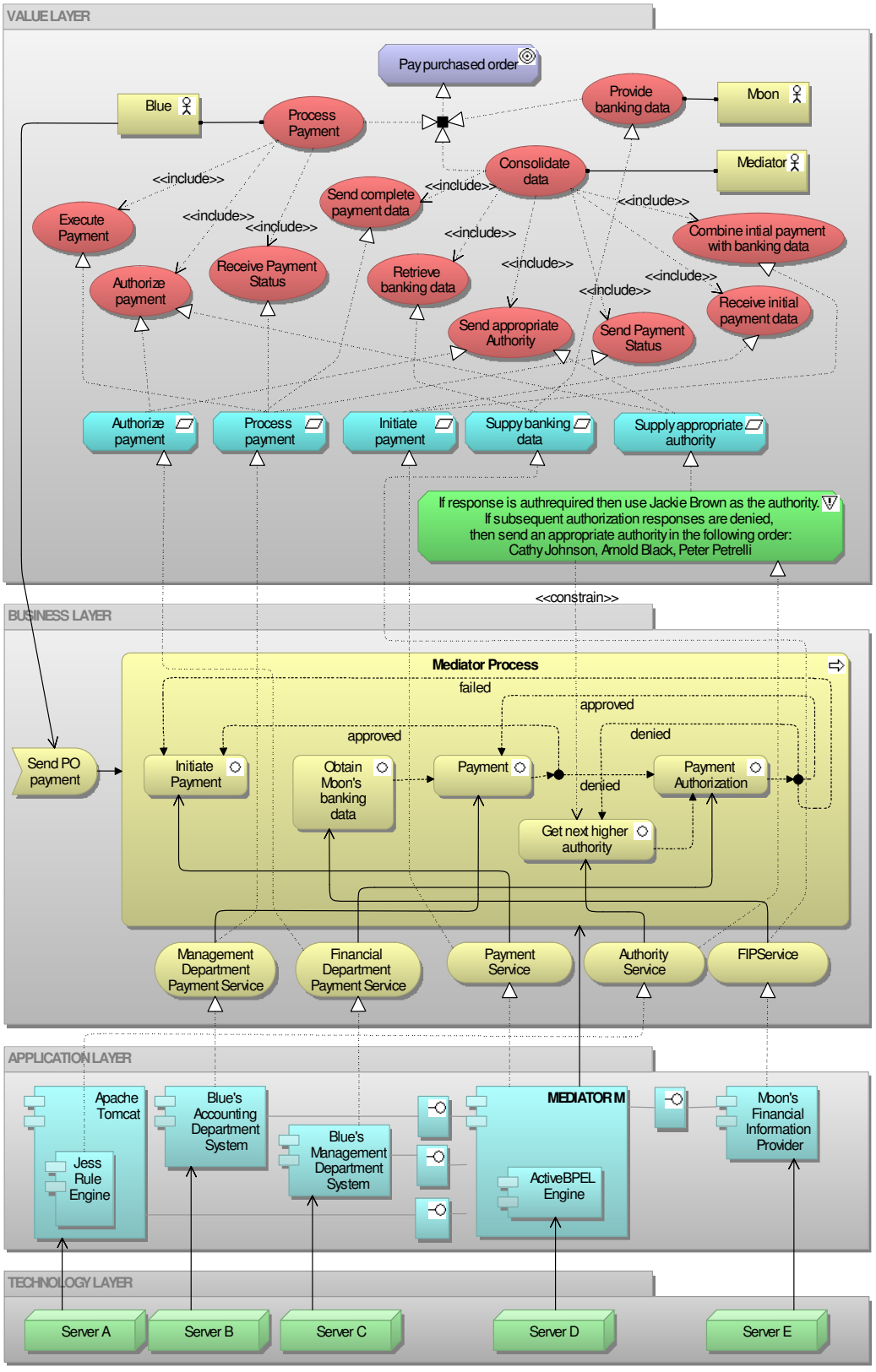
A

SWS Challenge Payment Problem Scenario solution artifacts

A.1. ARMOR goal model



A.2. ARMOR + ArchiMate enterprise architecture



A.3. Derived ACE sentences

Authority1:

```
If the response is authrequired
then the next authority is Jackie-Brown.
```

Authority2:

```
If the response is denied
and the previous authority is Jackie-Brown
then the next authority is Cathy-Johnson.
```

Authority3:

```
If the response is denied
and the previous authority is Cathy-Johnson
then the next authority is Arnold-Black.
```

Authority4:

```
If the response is denied
and the previous authority is Arnold-Black
then the next authority is Peter-Petrelli.
```

A.4. Generated RuleML files

Authority1:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleML>
    <oid>
        <Ind>
            Generated message from ACE text
            "If the response is a:authrequired
            then the next authority is p:Jackie-Brown."
            []
               [A, B, C]
               property(A, authrequired, pos)-1/7
               predicate(B, be, C, A)-1/4
               object(C, response, countable, na, eq, 1)-1/3
                   =>
                   [D, E]
                   property(E, next, pos)-1/10
                   predicate(D, be, E, named(Jackie-Brown))-1/12
                   object(E, authority, countable, na, eq, 1)-1/11
        </Ind>
    </oid>
    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>Authority1</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind/>
        </sender>
        <content>
            <Forall>
```

```
                <Var>A</Var>
                <Var>B</Var>
                <Var>C</Var>
                <Implies>
                    <And>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>A</Var>
                            <Ind>authrequired</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>B</Var>
                            <Ind>be</Ind>
                            <Var>C</Var>
                            <Var>A</Var>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>C</Var>
                            <Ind>response</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
                        </Atom>
                    </And>
                    <Exists>
                        <Var>D</Var>
                        <Var>E</Var>
                        <And>
                            <Atom>
                                <Rel>predicate</Rel>
                                <Var>D</Var>
                                <Ind>be</Ind>
                                <Var>E</Var>
                                <Var>named('Jackie-Brown')</Var>
                            </Atom>
                            <Atom>
                                <Rel>property</Rel>
                                <Var>G</Var>
                                <Ind>next</Ind>
                                <Ind>pos</Ind>
                            </Atom>
                            <Atom>
                                <Rel>object</Rel>
                                <Var>E</Var>
                                <Ind>authority</Ind>
                                <Ind>countable</Ind>
                                <Ind>na</Ind>
                                <Ind>eq</Ind>
                                <Data>1</Data>
                            </Atom>
                        </And>
                    </Exists>
                </Implies>
            </Forall>
        </content>
    </Message>
</RuleML>
```

Authority2:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleML>
    <oid>
        <Ind>Generated message from ACE text "If the response is denied
            and the previous authority is p:Jackie-Brown then the next
            authority is p:Cathy-Johnson.".
            []
               [A, B, C, D, E]
               property(A, denied, pos)-1/5
               predicate(B, be, C, A)-1/4
               object(C, response, countable, na, eq, 1)-1/3

               property(E, previous, pos)-1/8
               predicate(D, be, E, named(Jackie-Brown))-1/10
               object(E, authority, countable, na, eq, 1)-1/9
                   =>
                   [F, G]
                   property(G, next, pos)-1/16
                   predicate(F, be, G, named(Cathy-Johnson))-1/18
                   object(G, authority, countable, na, eq, 1)-1/17
        </Ind>
    </oid>
    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>Authority2</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind/>
        </sender>
        <content>
            <Forall>
                <Var>A</Var>
                <Var>B</Var>
                <Var>C</Var>
                <Var>D</Var>
                <Var>E</Var>
                <Implies>
                    <And>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>A</Var>
                            <Ind>denied</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>B</Var>
                            <Ind>be</Ind>
                            <Var>C</Var>
                            <Var>A</Var>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>C</Var>
                            <Ind>response</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
```

```
                        </Atom>
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>D</Var>
                            <Ind>be</Ind>
                            <Var>E</Var>
                            <Var>named('Jackie-Brown')</Var>
                        </Atom>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>E</Var>
                            <Ind>previous</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>E</Var>
                            <Ind>authority</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
                        </Atom>
                    </And>
                    <Exists>
                        <Var>F</Var>
                        <Var>G</Var>
                        <And>
                            <Atom>
                                <Rel>predicate</Rel>
                                <Var>F</Var>
                                <Ind>be</Ind>
                                <Var>G</Var>
                                <Var>named('Cathy-Johnson')</Var>
                            </Atom>
                            <Atom>
                                <Rel>property</Rel>
                                <Var>G</Var>
                                <Ind>next</Ind>
                                <Ind>pos</Ind>
                            </Atom>
                            <Atom>
                                <Rel>object</Rel>
                                <Var>G</Var>
                                <Ind>authority</Ind>
                                <Ind>countable</Ind>
                                <Ind>na</Ind>
                                <Ind>eq</Ind>
                                <Data>1</Data>
                            </Atom>
                        </And>
                    </Exists>
                </Implies>
            </Forall>
        </content>
    </Message>
</RuleML>
```

Authority3:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleML>
    <oid>
        <Ind>
            Generated message from ACE text "If the response is denied
            and the previous authority is p:Cathy-Johnson then the next
            authority is p:Arnold-Black.".
            []
               [A, B, C, D, E]
               property(A, denied, pos)-1/5
               predicate(B, be, C, A)-1/4
               object(C, response, countable, na, eq, 1)-1/3

               property(E, previous, pos)-1/8
               predicate(D, be, E, named(Cathy-Johnson))-1/10
               object(E, authority, countable, na, eq, 1)-1/9
                   =>
                   [F, G]
                   property(G, next, pos)-1/16
                   predicate(F, be, G, named(Arnold-Black))-1/18
                   object(G, authority, countable, na, eq, 1)-1/17
             </Ind>
    </oid>
    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>Authority3</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind/>
        </sender>
        <content>
            <Forall>
                <Var>A</Var>
                <Var>B</Var>
                <Var>C</Var>
                <Var>D</Var>
                <Var>E</Var>
                <Implies>
                    <And>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>A</Var>
                            <Ind>denied</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>B</Var>
                            <Ind>be</Ind>
                            <Var>C</Var>
                            <Var>A</Var>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>C</Var>
                            <Ind>response</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
                        </Atom>
```

```
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>D</Var>
                            <Ind>be</Ind>
                            <Var>E</Var>
                            <Var>named('Cathy-Johnson')</Var>
                        </Atom>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>E</Var>
                            <Ind>previous</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>E</Var>
                            <Ind>authority</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
                        </Atom>
                    </And>
                    <Exists>
                        <Var>F</Var>
                        <Var>G</Var>
                        <And>
                            <Atom>
                                <Rel>predicate</Rel>
                                <Var>F</Var>
                                <Ind>be</Ind>
                                <Var>G</Var>
                                <Var>named('Arnold-Black')</Var>
                            </Atom>
                            <Atom>
                                <Rel>property</Rel>
                                <Var>G</Var>
                                <Ind>next</Ind>
                                <Ind>pos</Ind>
                            </Atom>
                            <Atom>
                                <Rel>object</Rel>
                                <Var>G</Var>
                                <Ind>authority</Ind>
                                <Ind>countable</Ind>
                                <Ind>na</Ind>
                                <Ind>eq</Ind>
                                <Data>1</Data>
                            </Atom>
                        </And>
                    </Exists>
                </Implies>
            </Forall>
        </content>
    </Message>
</RuleML>
```

Authority4:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleML>
    <oid>
        <Ind>
            Generated message from ACE text "If the response is denied
            and the previous authority is p:Arnold-Black then the next
            authority is p:Peter-Petrelli .".
            []
               [A, B, C, D, E]
               property(A, denied, pos)-1/5
               predicate(B, be, C, A)-1/4
               object(C, response, countable, na, eq, 1)-1/3

               property(E, previous, pos)-1/8
               predicate(D, be, E, named(Arnold-Black))-1/10
               object(E, authority, countable, na, eq, 1)-1/9
                   =>
                   [F, G]
                   property(G, next, pos)-1/16
                   predicate(F, be, G, named(Peter-Petrelli))-1/18
                   object(G, authority, countable, na, eq, 1)-1/17
        </Ind>
    </oid>
    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>Authority4</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind/>
        </sender>
        <content>
            <Forall>
                <Var>A</Var>
                <Var>B</Var>
                <Var>C</Var>
                <Var>D</Var>
                <Var>E</Var>
                <Implies>
                    <And>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>A</Var>
                            <Ind>denied</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>B</Var>
                            <Ind>be</Ind>
                            <Var>C</Var>
                            <Var>A</Var>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>C</Var>
                            <Ind>response</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
```

```
                        </Atom>
                        <Atom>
                            <Rel>predicate</Rel>
                            <Var>D</Var>
                            <Ind>be</Ind>
                            <Var>E</Var>
                            <Var>named('Arnold-Black')</Var>
                        </Atom>
                        <Atom>
                            <Rel>property</Rel>
                            <Var>E</Var>
                            <Ind>previous</Ind>
                            <Ind>pos</Ind>
                        </Atom>
                        <Atom>
                            <Rel>object</Rel>
                            <Var>E</Var>
                            <Ind>authority</Ind>
                            <Ind>countable</Ind>
                            <Ind>na</Ind>
                            <Ind>eq</Ind>
                            <Data>1</Data>
                        </Atom>
                    </And>
                    <Exists>
                        <Var>F</Var>
                        <Var>G</Var>
                        <And>
                            <Atom>
                                <Rel>predicate</Rel>
                                <Var>F</Var>
                                <Ind>be</Ind>
                                <Var>G</Var>
                                <Var>named('Peter-Petrelli')</Var>
                            </Atom>
                            <Atom>
                                <Rel>property</Rel>
                                <Var>G</Var>
                                <Ind>next</Ind>
                                <Ind>pos</Ind>
                            </Atom>
                            <Atom>
                                <Rel>object</Rel>
                                <Var>G</Var>
                                <Ind>authority</Ind>
                                <Ind>countable</Ind>
                                <Ind>na</Ind>
                                <Ind>eq</Ind>
                                <Data>1</Data>
                            </Atom>
                        </And>
                    </Exists>
                </Implies>
            </Forall>
        </content>
    </Message>
</RuleML>
```

A.5. RuleML to JESS XSLT stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
================================================================================
RuleML2Jess stylesheet for RuleML v.0.91 by Cams Asuncion
Based on the work of Said Tabet at http://home.comcast.net/~stabet/page3.html
Runs on SAXON - The XSLT and XQuery Processor http://saxon.sourceforge.net/
================================================================================
-->
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">

    <xsl:output method="text" />

    <xsl:template match="RuleML">
        <xsl:apply-templates select="Message" />
    </xsl:template>

    <xsl:template match="Message">
        <xsl:apply-templates select="oid" />
        <xsl:apply-templates select="content" />
    </xsl:template>

    <xsl:template match="oid">
        <xsl:text>(defrule </xsl:text>
        <xsl:apply-templates select="Ind" />
        <xsl:text>&#x20;</xsl:text>
        <xsl:text>&#xa;</xsl:text>
        <xsl:text>&#x9;</xsl:text>
    </xsl:template>

    <xsl:template match="content">
        <xsl:apply-templates select="Forall" />
        <xsl:text>)</xsl:text>
    </xsl:template>

    <xsl:template match="Forall">
        <xsl:apply-templates select="Implies" />
    </xsl:template>

    <xsl:template match="Implies">
        <xsl:apply-templates select="And" />
        <xsl:text>
        <![CDATA[=>]]>
        </xsl:text>
        <xsl:text>(assert </xsl:text>
        <xsl:apply-templates select="Exists" />
        <xsl:text>)</xsl:text>
    </xsl:template>

    <xsl:template match="Exists">
        <xsl:apply-templates select="And" />
    </xsl:template>

    <xsl:template match="And">

        <xsl:variable name="quote">'</xsl:variable>
        <xsl:variable name="dash">-</xsl:variable>

        <xsl:for-each select="Atom">

            <!-- reorder atoms so that we start parsing
                 from the bottom (object) first -->
            <xsl:sort select="position()"
```

```
65              data-type="number" order="descending" />
66
67              <!-- store the value of Rel:
68                   which could be object | property | predicate -->
69              <xsl:variable name="RelValue" select="Rel" />
70
71              <xsl:choose>
72                  <!-- parse the first Ind of object -->
73                  <xsl:when test="$RelValue = 'object'">
74                      <xsl:text>(</xsl:text>
75                      <xsl:value-of select="Ind[position() = 1]" />
76                  </xsl:when>
77
78                  <!-- parse the third var of predicate
79                       tokenize into strings
80                       if there are two tokens then
81                          parse first token as first name of authority
82                          parse second token is second name of authority
83                       else
84                          don't parse -->
85                  <xsl:when test="$RelValue = 'predicate'">
86                      <xsl:text>&#x20;</xsl:text>
87                      <xsl:variable name="predicateString"
88                      select="Var[position() = 3]" />
89                      <xsl:variable name="tokens"
90                      select="tokenize($predicateString, $quote)" />
91                      <xsl:if test="count($tokens)">
92                          <xsl:for-each select="$tokens">
93                              <xsl:if test="position()=2">
94                                  <xsl:variable name="nameString" select="." />
95                                  <xsl:variable name="nameToken"
96                                  select="tokenize($nameString, $dash)" />
97                                  <xsl:for-each select="$nameToken">
98                                      <xsl:choose>
99                                          <xsl:when test="count($nameToken)">
100                                             <xsl:if test="position()=1">
101                                                 <xsl:value-of select="." />
102                                             </xsl:if>
103                                             <xsl:if test="position()=2">
104                                                 <!-- insert space -->
105                                                 <xsl:text>&#x20;</xsl:text>
106                                                 <xsl:value-of select="." />
107                                                 <xsl:text>)</xsl:text>
108                                                 <xsl:text>&#x20;</xsl:text>
109                                                 <xsl:text>&#xa;</xsl:text>
110                                                 <xsl:text>&#x9;</xsl:text>
111                                             </xsl:if>
112                                         </xsl:when>
113                                         <xsl:otherwise>
114                                             <xsl:value-of select="." />
115                                         </xsl:otherwise>
116                                     </xsl:choose>
117                                 </xsl:for-each>
118                             </xsl:if>
119                         </xsl:for-each>
120                     </xsl:if>
121                 </xsl:when>
122
123                 <!-- parse first ind of property
124                      only when it's a child of implies-->
125                     <xsl:when test="$RelValue = 'property'">
126                         <xsl:variable name="is-last" select="position() = last()
127                         <xsl:if test="$is-last">
128                             <xsl:value-of select="Ind[position() = 1]" />
129                             <xsl:text>)</xsl:text>
130                         </xsl:if>
131                 </xsl:when>
```

```
132             </xsl:choose>
133         </xsl:for-each>
134         <xsl:text> </xsl:text>
135     </xsl:template>
136 </xsl:stylesheet>
```

A.6. Generated JESS rules

Authority1:

```
1(defrule Authority1
2    (response authrequired)
3        =>
4        (assert (authority Jackie Brown)
5    ))
```

Authority2:

```
1 (defrule Authority2
2    (authority Jackie Brown)
3    (response denied)
4        =>
5        (assert (authority Cathy Johnson)
6    ))
```

Authority3:

```
1 (defrule Authority3
2    (authority Cathy Johnson)
3    (response denied)
4        =>
5        (assert (authority Arnold Black)
6    ))
```

Authority4:

```
1(defrule Authority4
2    (authority Arnold Black)
3    (response denied)
4        =>
5        (assert (authority Peter Petrelli)
6    ))
```

A.7. JESS rule set added with JESS-specific code

```
1 ; Assert to working memory the response returned
2 ; by authorize operation of Blue's Management
3 ; Depaertment System
4
5 (assert (response (fetch RESPONSE-CODE)))
6 (facts)
7
```

```
; If the response is authrequired
; then the next authority is Jackie-Brown.
(defrule Authority1
    ?code <- (response authrequired)
     =>
        (printout t "It's Authority1" crlf)
        (assert (authority Jackie Brown))
        (store AUTH-FIRST-NAME Jackie)
        (store AUTH-LAST-NAME Brown)
        (retract ?code)
        (facts)
)

; If the response is denied
; and the previous authority is Jackie-Brown
; then the next authority is Cathy-Johnson.
(defrule Authority2
    ?auth <- (authority Jackie Brown)
    ?code <- (response denied)
     =>
        (printout t "It's  Authority2" crlf)
        (assert (authority Cathy Johnson))
        (store AUTH-FIRST-NAME Cathy)
        (store AUTH-LAST-NAME Johnson)
        (retract ?code)
        (retract ?auth)
        (facts)
)

; If the response is denied
; and the previous authority is p:Cathy-Johnson
; then the next authority is p:Arnold-Black.
(defrule Authority3
    ?auth <- (authority Cathy Johnson)
    ?code <- (response denied)
    =>
        (printout t "It's Authority3" crlf)
        (assert (authority Arnold Black))
        (store AUTH-FIRST-NAME Arnold)
        (store AUTH-LAST-NAME Black)
        (retract ?code)
        (retract ?auth)
        (facts)
)

; If the response is denied
; and the previous authority is p:Arnold-Black
; then the next authority is p:Peter-Petrelli.
(defrule Authority4
    ?auth <- (authority Arnold Black)
    ?code <- (response denied)
    =>
        (printout t "It's Authority4" crlf)
        (assert (authority Peter Petrelli))
        (store AUTH-FIRST-NAME Peter)
        (store AUTH-LAST-NAME Petrelli)
        (retract ?code)
        (retract ?auth)
        (facts)
)
```

```
68
69 ; retract leftover facts
70 (defrule clear-last-authority
71     ?code <- (response denied)
72     ?auth <- (authority Peter Petrelli)
73     =>
74         (printout t "Cleared!" crlf)
75         (retract ?auth)
76         (retract ?code)
77         (store AUTH-FIRST-NAME "")
78         (store AUTH-LAST-NAME "")
79         (facts)
80 )
81
82 ; retract (reponse denied) fact
83 ; if no other authority facts are
84 ; asserted into the working memory
85 (defrule clear-denied-only
86     ?code <- (response denied)
87     =>
88         (printout t "Cleared!" crlf)
89         (retract ?code)
90         (store AUTH-FIRST-NAME "")
91         (store AUTH-LAST-NAME "")
92         (facts)
93 )
```

A.8. WSDL for AuthorityService

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3     xmlns:apachesoap="http://xml.apache.org/xml-soap"
4     xmlns:impl="authority"
5     xmlns:ns="http://schemas.xmlsoap.org/soap/encoding/"
6     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7     xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
8     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9     name="AuthorityService"
10    targetNamespace="authority">
11    <wsdl:types>
12        <xsd:schema xmlns="authority"
13            elementFormDefault="qualified"
14            targetNamespace="authority">
15            <xsd:element name="AuthorityRequest">
16                <xsd:complexType>
17                    <xsd:sequence>
18                        <xsd:element name="responseCode"
19                            nillable="false"
20                            type="xsd:string" />
21                    </xsd:sequence>
22                </xsd:complexType>
23            </xsd:element>
24            <xsd:element name="AuthorityResponse">
25                <xsd:complexType>
26                    <xsd:sequence>
27                        <xsd:element name="firstName"
28                        nillable="false" type="xsd:string" />
29                        <xsd:element name="lastName"
30                        nillable="false" type="xsd:string" />
```

```
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="AuthorityRequestMessage">
        <wsdl:part element="impl:AuthorityRequest"
        name="AuthorityRequest" />
    </wsdl:message>
    <wsdl:message name="AuthorityResponseMessage">
        <wsdl:part element="impl:AuthorityResponse"
        name="AuthorityResponse" />
    </wsdl:message>
    <wsdl:portType name="AuthorityServicePortType">
        <wsdl:operation name="getNextAuthority">
            <wsdl:input
            message="impl:AuthorityRequestMessage" />
            <wsdl:output
            message="impl:AuthorityResponseMessage" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="AuthorityServiceSoapBinding"
        type="impl:AuthorityServicePortType">
        <wsdlsoap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="getNextAuthority">
            <wsdlsoap:operation soapAction="getNextAuthority"
                style="document" />
            <wsdl:input>
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="AuthorityService">
        <wsdl:port binding="impl:AuthorityServiceSoapBinding"
        name="AuthorityService">
        <wsdlsoap:address
        location="http://localhost:8080/sws/services/AuthorityService"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

A.9. Java wrapper for getNextAuthority operation of AuthorityService

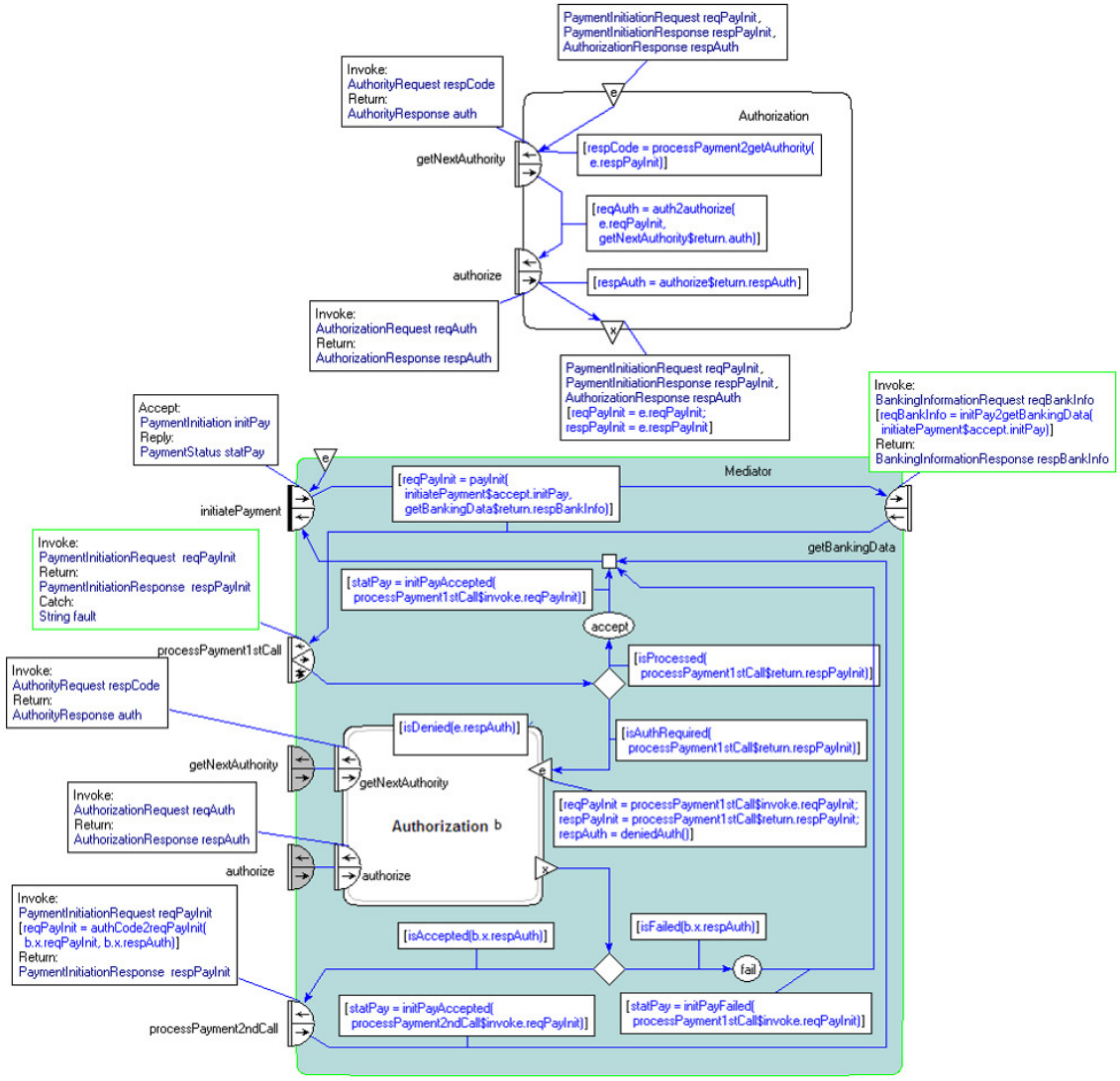
```
 * AuthorityServiceSoapBindingImpl.java
package authority;
import java.rmi.RemoteException;

/**
 * Web service wrapper to invoke Jess Rule
 * @author Cams Asuncion
 */
```

```
 9public class AuthorityServiceSoapBindingImpl
10        implements authority.AuthorityServicePortType {
11
12    private static Rete _engine;
13
14    //create singleton instance of the Jess engine
15    public static synchronized Rete getEngineInstance()
16            throws JessException{
17        if (_engine == null) {
18            _engine = new Rete();
19            //clear facts and rules from working memory
20            _engine.clear();
21        }
22        return _engine;
23    }
24
25    public AuthorityResponse getNextAuthority(
26            AuthorityRequest req)
27        throws java.rmi.RemoteException {
28
29        if (req == null){
30            throw new IllegalArgumentException();
31        }else if (req.getResponseCode() == null){
32            throw new IllegalArgumentException();
33        }
34
35        AuthorityResponse resp = new AuthorityResponse();
36        resp.setFirstName("");
37        resp.setLastName("");
38
39        String respCode = req.getResponseCode().toLowerCase().trim();
40        System.out.println("req.getResponseCode(): " + respCode);
41
42        try {
43
44            String firstName = "";
45            String lastName = "";
46
47            getEngineInstance().store("RESPONSE-CODE",
48                    new Value(respCode, RU.SYMBOL));
49            getEngineInstance().batch("sws-jess-gen.v2.clp");
50            getEngineInstance().run();
51            Value authFirstName = getEngineInstance()
52                                        .fetch("AUTH-FIRST-NAME");
53            Value authLastName = getEngineInstance()
54                                        .fetch("AUTH-LAST-NAME");
55
56            //extract authority from context
57            if (authFirstName != null
58                    & authLastName != null){
59
60                firstName = authFirstName.stringValue(
61                            getEngineInstance().getGlobalContext());
62                lastName = authLastName.stringValue(
63                            getEngineInstance().getGlobalContext());
64            }
65
66            resp.setFirstName(firstName);
67            resp.setLastName(lastName);
68
```

```
69              System.out.println("resp.getFirstName(): "
70                      + resp.getFirstName());
71              System.out.println("resp.getLastName(): "
72                      + resp.getLastName());
73
74          } catch (JessException je) {
75              je.printStackTrace();
76          } catch (Exception e){
77              e.printStackTrace();
78          }
79          return resp;
80      }
81  }
```

A.10. ISDL behavior model



A.11. MDSL semantic data mapping

```
transformation sws_payment_problem_scenario{
    namespaces{
        authority   = "authority";
        moon        = "mooncompany";
        fd          = "org.sws_challenge.schemas.financialdepartment";
        iso         = "org.sws_challenge.schemas.iso20022";
        md          = "org.sws_challenge.schemas.managementdepartment";
        util        = "java.util";
        lang        = "java.lang";
        fn          = "";
    }

    mapping initPay2getBankingData {
        target moon:BankingInformationRequest reqBankInfo {
            requestId = "requestId";
        }
        expressions{
            requestId = "token_1234567890";
        }
    }

    mapping payInit{
        target fd:PaymentInitiationRequest reqPayInit {
            swiftCode       = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgt/finInstnId/bic";
            currencyCode    = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgtAcct/ccy";
            accountNumber   = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgtAcct/id/iban";
            cd              = "pain00100102/pmtInf/cdtTrfTxInf/cdtrAgtAcct/tp/cd";
        }
        source moon:PaymentInitiation initPay{
            reqPayInit      = "pain00100102";
        }
        source moon:BankingInformationResponse respBankInfo{
            swiftCode       = "swiftCode";
            currencyCode    = "currencyCode";
            accountNumber   = "accountNumber";
        }
        expressions{
            cd              = "CACC";
        }
    }

    mapping isProcessed{
        target lang:Boolean result {
        }
        source fd:PaymentInitiationResponse  respPayInit{
            status          = "paymentInitiationResponseCode";
        }
        expressions {
            processedStatus = "PROCESSED";
            result          = fn:equalsTo(status, processedStatus);
        }
    }

    mapping isAuthRequired{
        target lang:Boolean result {
        }
        source fd:PaymentInitiationResponse  respPayInit{
            status              = "paymentInitiationResponseCode";
        }
        expressions {
            authRequiredStatus  = "AUTHREQUIRED";
            result              = fn:equalsTo(status, authRequiredStatus);
        }
    }

    mapping initPayAccepted{
        target moon:PaymentStatus iStatPay{
            status      = "paymentStatusCode/";
            msgId       = "pain00200102/grpHdr/msgId";
```

```
70              creditDtTm  = "pain00200102/grpHdr/creDtTm";
71              debitTr     = "pain00200102/grpHdr/initgPty";
72              debitTrAgt  = "pain00200102/grpHdr/dbtrAgt";
73              groupHeader = "pain00200102/orgnlGrpInfAndSts";
74              grpSts      = "pain00200102/orgnlGrpInfAndSts/grpSts";
75          }
76          source fd:PaymentInitiationRequest reqPayInit{
77              groupHeader = "pain00100102/grpHdr";
78              msgId       = "pain00100102/grpHdr/msgId";
79              creditDtTm  = "pain00100102/grpHdr/creDtTm";
80              debitTr     = "pain00100102/pmtInf/dbtr";
81              debitTrAgt  = "pain00100102/pmtInf/dbtrAgt";
82          }
83          expressions{
84              status = "PI_ACCEPTED";
85              grpSts = "ACCP";
86          }
87      }
88
89      mapping processPayment2getAuthority{
90          target authority:AuthorityRequest respCode {
91              status  = "responseCode";
92          }
93          source fd:PaymentInitiationResponse respPayInit {
94              status  = "paymentInitiationResponseCode";
95          }
96      }
97
98      mapping auth2authorize{
99          target md:AuthorizationRequest reqAuth{
100             firstName   = "authority/firstName";
101             lastName    = "authority/lastName";
102             pain        = "pain00100102";
103         }
104         source fd:PaymentInitiationRequest reqPayInit{
105             pain        = "pain00100102";
106         }
107         source authority:AuthorityResponse auth{
108             firstName   = "firstName";
109             lastName    = "lastName";
110         }
111     }
112
113     mapping authCode2reqPayInit{
114         target fd:PaymentInitiationRequest reqPayInit{
115             authRespCode    = "authorizationCode";
116             pain            = "pain00100102";
117         }
118         source fd:PaymentInitiationRequest reqPayInit{
119             pain            = "pain00100102";
120         }
121         source md:AuthorizationResponse authResp{
122             authRespCode    = "authorizationCode";
123         }
124     }
125
126     mapping isAccepted{
127         target lang:Boolean result {
128         }
129         source md:AuthorizationResponse authResp{
130             respCode    = "responseCode";
131         }
132         expressions {
133             accepted    = "ACCEPTED";
134             result      = fn:equalsTo(respCode, accepted);
135         }
136     }
137
138     mapping isDenied{
139         target lang:Boolean result {
140         }
141         source md:AuthorizationResponse authResp{
```

```
141         source md:AuthorizationResponse authResp{
142             respCode    = "responseCode";
143         }
144         expressions {
145             accepted    = "DENIED";
146             result      = fn:equalsTo(respCode, accepted);
147         }
148     }
149
150     mapping isFailed{
151         target lang:Boolean result {
152         }
153         source md:AuthorizationResponse authResp{
154             respCode    = "responseCode";
155         }
156         expressions {
157             accepted    = "FAILED";
158             result      = fn:equalsTo(respCode, accepted);
159         }
160     }
161
162     mapping initPayFailed{
163         target moon:PaymentStatus iStatPay{
164             status      = "paymentStatusCode";
165             msgId       = "pain00200102/grpHdr/msgId";
166             creditDtTm  = "pain00200102/grpHdr/creDtTm";
167             debitTr     = "pain00200102/grpHdr/initgPty";
168             debitTrAgt  = "pain00200102/grpHdr/dbtrAgt";
169             groupHeader = "pain00200102/orgnlGrpInfAndSts";
170             grpSts      = "pain00200102/orgnlGrpInfAndSts/grpSts";
171         }
172         source fd:PaymentInitiationRequest reqPayInit{
173             groupHeader = "pain00100102/grpHdr";
174             msgId       = "pain00100102/grpHdr/msgId";
175             creditDtTm  = "pain00100102/grpHdr/creDtTm";
176             debitTr     = "pain00100102/pmtInf/dbtr";
177             debitTrAgt  = "pain00100102/pmtInf/dbtrAgt";
178         }
179         expressions{
180             status = "PI_REFUSED_AUTH_FAILED";
181             grpSts = "RJCT";
182         }
183     }
184 }
```

B

Discourse Representation Structures

B.1. Overview

Discourse Representation Structure (DRS) is a formal method of dealing with contextual meaning across multiple sentences. DRS is based on Discourse Representation Theory (DRT) proposed by Hans Kamp in his study of the dynamic interpretation of natural language (Kamp and Reyle, 1990). The idea is to interpret a natural language discourse (meaning the sequence of sentences spoken by a speaker) in the context of a representation structure (van Eijck, 2005). One of the main objectives of DRS is to resolve ambiguities in anaphoric references. A simple example is given by van Eijck (2005):

A man met an attractive woman. He smiled at her. (1)

To resolve the link between the anaphoric pronouns (i.e. "*he*" and "*her*" in the second sentence) and their antecedents (i.e. "*man*" and "*woman*" in the first sentence, respectively) DRS approaches this by translating such pronouns as variables bound by their antecedents. To do this, the previous discourse in example (1) can be viewed as:

A man_1 met an attractive woman_2. He_1 smiled at her_2. (2)

A conventional way of representing DRS is through the box notation. Consider the example given by Hirtle (2006):

The owner separates the cat from the dog. It growls. (3)

The equivalent box notation of example (3) in DRS would thus look like the one in Figure B-1. Here, we can identify that the one who growls is the "*dog*" and not the owner as both are bound to the same variable *D*. The variables *F*, *B*, and *D* are called *referents*. Below the referents is the list of *conditions* (or logical atoms).

F B D
owner(F)
cat(B)
dog(D)
separate(F,B,D)
growl(D)

Figure B-1: DRS box notation (Hirtle, 2006)

B.2. DRS in ACE

ACE uses DRS to make ACE more interoperable with other technologies such as OWL-DL. ACE has a slightly different representation of the box notation in DRS. Consider the example given in Figure B-2. ACE uses predefined conditions such as object, property, and predicate. The `[]` represent universal quantification while `=>` implies logical implication. The `[]` part usually corresponds to the `IF` part of a rule, while the `=>` takes the `THEN` part of the rule.

Every company that buys at least 2 standard machines gets a discount.

```
[]
   [A, B, C]
   object(A, atomic, company, object, cardinality, count_unit, eq, 1)-1
   object(B, group, machine, object, cardinality, count_unit, geq, 2)-1
   property(B, standard)-1
   predicate(C, event, buy, A, B)-1
   =>
   [D, E]
   object(D, atomic, discount, object, cardinality, count_unit, eq, 1)-1
   predicate(E, event, get, A, D)-1
```

Figure B-2: DRS pretty print in ACE (Fuchs and Kaljurand, 2006)

The following figures provide information to the details of the predicate conditions.

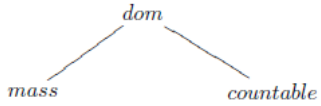
The `object`-predicates stand for objects that are introduced by the different forms of nouns.

```
object(Ref,Noun,Quant,Unit,Op,Count)
```

`Ref` The variable that stands for this object and that is used for references.

`Noun` The noun (mass or countable) that was used to introduce the object.

`Quant` This is one of `{dom,mass,countable}` and defines the quantisation of the object. The tree structure below shows the hierarchy of these values.



`Unit` If the object was introduced together with a measurement noun (e.g. "2 kg of apples") then this entry contains the value of the measurement noun (e.g. `kg`). Otherwise, this entry is `na`.

`Op` One of `{eq,geq,greater,exactly,na}`. `eq` stands for "equal" and `geq` for "greater or equal". Note that `leq` and `less` can not appear here but only in the `quantity`-predicate.

`Count` A positive number or `na`. Together with `Unit` and `Op`, this defines the cardinality or extent of the object.

Figure B-3: `object`-predicates representation in ACE (Fuch, et al., 2009)

The `predicate`-predicates stand for relations that are introduced by intransitive, transitive, and ditransitive verbs.

intransitive	`predicate(Ref,Verb,SubjRef)`
transitive	`predicate(Ref,Verb,SubjRef,ObjRef)`
ditransitive	`predicate(Ref,Verb,SubjRef,ObjRef,IndObjRef)`

`Ref` A variable that stands for this relation and that is used to attach modifiers (i.e. adverbs and prepositional phrases).

`Verb` The intransitive, transitive, or ditransitive verb.

`SubjRef` A variable or expression that stands for the subject.

`ObjRef` A variable or expression that stands for the direct object.

`IndObjRef` A variable or expression that stands for the indirect object.

Figure B-4: `predicate`-predicates representation in ACE (Fuch, et al., 2009)

The `property`-predicates stand for properties that are introduced by adjectives. The references can either be variables or expressions.

1-ary	`property(Ref1,Adjective,Degree)`
2-ary	`property(Ref1,Adjective,Degree,Ref2)`
3-ary	`property(Ref1,Adjective,Ref2,Degree,CompTarget,Ref3)`

`Ref1` The variable or expression that stands for the primary object of the property (i.e. the subject).

`Ref2` The variable or expression that stands for the secondary object of the property.

`Ref3` The variable or expression that stands for the tertiary object of the property.

`Adjective` The intransitive or transitive adjective.

`Degree` This is one of `{pos,pos_as,comp,comp_than,sup}` and it defines the degree of the adjective. Positive and comparative forms can have an additional comparison target ("as rich as ...", "richer than ..."), and for those cases `pos_as` and `comp_than` are used.

`CompTarget` This is one of `{subj,obj}` and it defines for transitive adjectives whether the comparison targets the subject ("John is more fond-of Mary than Bill") or the object ("John is more fond-of Mary than of Sue").

Figure B-5: `property`-predicates representation in ACE (Fuch, et al., 2009)

C

RuleML to Jess Transformation

C.1. Overview

Our approach to transforming RuleML to Jess is largely specific to the SWS Challenge. As More work still needs to be done to make the transformation more generic and reusable. Our transformation is written in XSLT where it takes the RuleML output of ACE2RRML as its input. The XSLT stylesheet is simply parses the given RuleML XML file, picks out relevant information and construct the equivalent in rule in Jess. This chapter discusses how we designed the transformation. The complete stylesheet is found in Appendix A.5.

For the SWS Challenge, we have broken down the business rule modeled in ARMOR into four distinct and more detailed ACE sentences. Each ACE sentence is given its equivalent translation to RuleML. Similarly, we also translate each RuleML authority rule one at a time. But, we have made it a point to create just one XSLT stylesheet for all four RuleML files.

C.2. Details

Take for example, the ACE rule `Authority1`:

```
Authority1:

If the response is authrequired
then the next authority is Jackie-Brown.
```

Figure C-1: `Authority1` in ACE

The equivalent DRS representation of the ACE `Authority1` rule is thus:

```
[]
   [A, B, C]
   property(A, authrequired, pos)-1/7
   predicate(B, be, C, A)-1/4
   object(C, response, countable, na, eq, 1)-1/3
       =>
       [D, E]
       property(E, next, pos)-1/10
       predicate(D, be, E, named(Jackie-Brown))-1/12
       object(E, authority, countable, na, eq, 1)-1/11
```

Figure C-2: `Authority1` in DRS

Notice that all subjects of the IF and THEN parts are represented in DRS as objects. Verbs on the other hand are treated as predicates. Notice, however, that the Jackie Brown treated as a direct object of the transitive verb "is" (cf. Figure B-4). DRS represents Jackie Brown as a proper name by enclosing it within the "named()" construct while connecting the names with a dash. Adjectives are represented as properties. For example, the "authrequired" acts as an adjective complement to the IF part of the rule so that it is represented as a property in DRS.

A rule in Jess is defined using the defrule construct. This is inserted automatically by the XSLT stylesheet while using appropriate closing and opening parentheses. The conversation ID that was used during the transformation of ACE to RuleML using ACE2RRML will be used as the name of the rule. In this case, the ID is Authority1.

A defrule in Jess is divided into two parts: the *pattern* and the *action*. All atoms in the enclosing < And > element under the <Implies> element will be used as a pattern in Jess (before the => symbol). Thus, an <And> element constitutes one pattern. On the other hand, all the atoms in the enclosing < And > element under the <Exists> element will be treated as actions in Jess (after the => symbol). Thus one action corresponds to one < And > element. Furthermore, when an < And > element is to be interpreted as an action, the XSLT stylesheet inserts the assert construct before the list.

We represent DRS objects as the head of a list in Jess (i.e. the first word that appears in a pattern or action). For example, in the IF part of Authority1 shown in Figure C-2, the object is response, thus in Jess it appears as (response…).

We make use of DRS predicates only when they can be tokenizable into two Strings. This ensures that we are able to get only the first and last names of the Authorities and not the predicate "be".

To keep our transformation simpler, we opted not to include the DRS property (e.g next and previous) translations in the Jess rule except when the property belongs to the first <And> element under the <Implies> which allows us to add the properties denied and authrequired.

All in all, our XSLT transformation still requires some improvement to make it more flexible and reusable. Our translation is rather literal, a more logical approach may be more appropriate. We have not tested our stylesheet with other forms of rule structures in ACE.

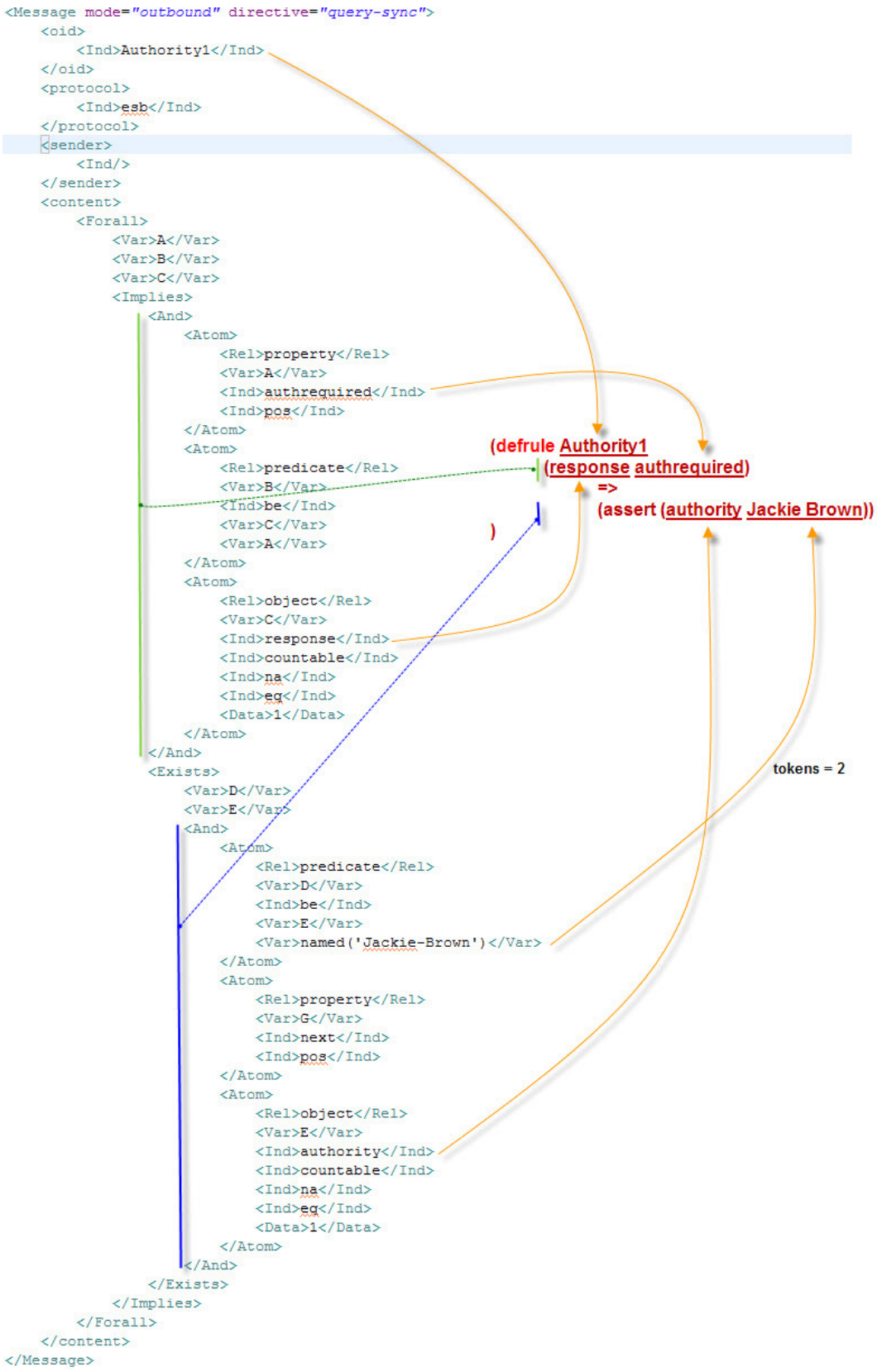


Figure C-3: RuleML to Jess transformation