Towards a Model-Driven Approach to Support SOA-Based Web-Business Platforms

THESIS

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

By

HARSHAVARDHAN JEGADEESAN

Under the Supervision of

Ramana Polavarapu, Ph.D

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN) INDIA

2009

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN) INDIA

CERTIFICATE

This is to certify that the thesis entitled "**Towards a Model-Driven Approach to Support SOA-Based Web-Business Platforms**" and submitted by **Mr. Harshavardhan Jegadeesan** ID. No. **2004PHXF431** for the award of Ph.D degree of the institute embodies original work done by him under my supervision.

_____

Signature of Supervision

(Ramana Polavarapu, Ph.D)

Date: January 18, 2009

Place: Bangalore, India

To my parents, with love and admiration!

Acknowledgements

I am immensely thankful to Prof. L. K. Maheshwari, Vice-Chancellor of BITS Pilani for providing me this opportunity to pursue the PhD program. I express my gratitude to Prof. Ravi Prakash, Dean, Research and Consultancy Division (RCD), BITS Pilani for his constant official support and encouragement. I thank Dr. Hemanth Jadav, Mr. Dinesh Kumar, Ms. Monica Sharma, Mr. Sharad Shrivastava, Mr. Gunjan Soni, Mr. Amit Kumar and Ms. Sunita Bansal, nucleus members of RCD, BITS Pilani for their cooperation and guidance. I also express my gratitude to the office staff of RCD whose secretarial assistance helped me in submitting the various evaluation documents in time and give pre-submission seminar smoothly. I thank my Doctoral Advisory Committee (DAC) members, Prof. Rahul Banerjee and Prof. Navneet Goyal, who spared their valuable time to go through my draft thesis and were audience to my pre-submission seminar in order to provide valuable suggestions.

This research and thesis would not have been possible, but for the support, encouragement and sometimes prodding of a lot of people. At first, I would like to thank all the souls who have influenced my thoughts and deeds, and from whom I have learnt all my life. My sincere thanks to my supervisor Dr. Ramana Polavarapu who allowed me to think and work independently, providing me the necessary guidance to bring focus and rigor to my research. Interacting with him has always been informative, motivating and highly productive.

I would like to thank Prof. Sundar Balasubramanian, who actually inspired me to pursue this research work. Working with him, I have imbibed abstract thinking and problem solving skills which have proved really helpful in my career. He is the best mentor one could possibly have. I would also like to express heart-felt gratitude to Prof. Rahul Banerjee, my guru, who instilled confidence in me, stood by me, trusted my capabilities and allowed me to grow as an individual. I would also like to sincerely thank Prof.

Pramod Chandra P. Bhatt for providing me guidance on my research and constantly reviewing my work and providing valuable feedback.

My sincere thanks are to my employer, SAP. SAP provided me with a thought-provoking research environment, real-world research problems and a bunch of really talented individuals to work with. I would like to specially thank Ramakrishna Yarlapati, Senior Vice-President - Business Suite, for supporting my research. Special thanks to David S. Frankel, SAP's Chief Standards Architect, for reviewing my metamodels and providing me deep insights in practical model-driven development. I would also like to thank Dr. Alistair Barros of SAP Research for his reviews and support. I would like to thank Yuvaraj Raghuvir, Chief Development Architect and a BITS alumnus for his pep talks which actually helped me complete my thesis. I would like to make a special mention of Blank Ink, our intrapreneurial venture (20% project) within SAP Labs and my co-founding engineers - Chandra, Tarun and Surabhi – it has been real pleasure working with them. All the real-world e-commerce scenarios in this thesis are straight out of Blank Ink's end-to-end internet selling platform. My special thanks are due to Nir Paikowsky, Senior Director of SME product strategy for shaping my understanding on online business platforms. My experience with Blank Ink, apart from lot of other things, gave me a good understanding of web-business platforms.

Beyond doubt this research would not have been possible without the strong support and encouragement of my parents. Thank you Mom and Dad, you are the best! My brother Ajay Jegadeesan has always been a pillar of strength. Last, but definitely not the least I appreciate the support of my wife Gayatri, without her tremendous support, inspiration and warmth it would not have been possible to manage both work as well as research. Gayatri, thanks, I love you dearly!

Abstract

The advent of the Internet has fostered commerce on the web, creating successful web businesses. More recently, web businesses which typically started out as websites are morphing into web-business platforms. They are building product platforms on which end-users can personalize their experiences and customize it for their needs. For example, eBay® provides an auctioning and internet selling platform which could be used by different consumers in different ways – an average Joe can auction used items out of his garage, an entrepreneur from India can sell Indian handicraft, gems and jewelry to customers in Europe and America, and a manufacturing company can auction its excess inventory in the online marketplace – all this with their own tailor-made, co-created user experiences. A *platform strategy* is getting increasingly critical for web-business owners to engage the community – an eco-system of developers, entrepreneurs and customers to co-innovate and create value around their web-business platforms.

How do platform-owners operationalize their platform strategy? The solution lies in "opening-up" their web-business platforms, thereby exposing their business capabilities to consumers in a way they can readily use them. This is quite similar to Windows® Application Programming Interfaces (APIs) which provide well-defined interfaces to developers to build applications on top of the Windows® operating systems. These APIs abstract the "internal workings" of platform functions, providing clean and easy to use interfaces to invoke these functions. In order to address a larger community, APIs for web-business platforms or web APIs as we call them have to be based-on interoperable open-standards. Predominantly, the web APIs are built using open-standards compliant webservices. By using these webservices, the community can build mash-ups by leveraging content from multiple information sources; entrepreneurs can exploit synergies between content and services from different providers to build end-to-end applications for customers.

Though the concept of “opening-up” web-business platforms through webservices seems pretty intuitive, there are certain critical technical issues which have to be addressed by the platform-owners, in order to operationalize it. For instance, webservice technologies and standards are in a state of constant flux. There are heterogeneities in design approaches, underlying invocation protocols and client-side consumption environments. Our research is motivated by these issues; the focus of our research and subsequently this thesis is to address these issues and facilitate “opening-up” of web-business platforms.

Our solution is based on a model-driven design and development approach in order to define service artifacts. This allows us to capture the solution space using high-level conceptual models, thus, delaying technology decisions to later stages of services development. We provide model views, metamodels and tools-compliant models to support services development in the context of web-business platforms. We also validate our research by applying it to non-trivial and real-world web-business platform scenarios.

Table of Contents

List of Figures

List of Tables

List of Conventions

1. Service names are written in small caps.
2. Service Interface names are suffixed with an (SI) superscript.
3. Service Operation names are suffixed with an (OP) superscript.
4. Service Exception names are suffixed with an (EX) superscript.
5. Message names are suffixed with an (M) superscript.
6. Service Interaction Point names are suffixed with an (IP) superscript.

Chapter 1

Introduction

The most recent trend among internet players such as eBay®, Amazon®, Force.com, Google® and many others is the "opening-up" of their software platforms [1]. These businesses are ceasing to be mere websites and are evolving into web-business platforms by pursuing a platform strategy [2-4]. They are opening up their business through the web, allowing their business functions to be accessed programmatically by a vibrant community - a community of developers and business partners. Doing so, they promote community-driven creation of value-added services and solutions for their customers faster than they could possibly create by themselves [5]. More often than not, the business functions of these web-business platforms are provided as open-standards compliant webservices. Technically, platform owners are service providers, providing their standard business functions as webservices based on service-oriented architecture principles. Each service represents an underlying business capability.

1.1 Problem Area

Every web-business platform owner would want to incrementally expose their business functions as externally accessible services based on stakeholders (the community as well as the customers) needs. While doing so, they face certain critical challenges. Although some of these challenges are inherent to the solution approach – service-oriented architectures, specifically the webservices technology. Nonetheless, these challenges need to be overcome [6].

- Currently, there is a rapid evolution of standards in the webservices technology space. Webservice technologies like WSDL [7] (for service description) and SOAP

[8] (for service invocation) have stabilized, however, associated specifications (WS-*) [9] are still evolving and are likely to result in more competing standards. These standards are promoted by different standards bodies and industry lobbies. In addition, alternate approaches such as Representational State Transfer (REST) [10] have created more heterogeneity in the services ecosystem. As the standards and underlying technologies evolve at a rapid pace, the longevity of the solutions built on them reduces. We call this the *Evolving Standards Problem*.

- Service Metadata is currently lean and incomplete. The WS-* standards which describe various facets of service metadata are semantically weak. For example, to access an eBay® webservice [11], the registration information (*developer key* and a *merchantID* obtained while signing up with the eBay® developer program) must be supplied for each service invocation. However, this information is not a part of the formal service description; instead it is specified in the developer documentation. Not all service facets can be adequately described by existing formal service description mechanisms; therefore automated ways of service consumption is still not a reality. We call this the *Lean Service Metadata Problem*.

- Services in a services marketplace have to be differentiated from that of competitor service offerings in order to sustain or gain market share. In essence, service offerings have to be offered to consumers at competitive terms than competing services in the marketplace. Competitive positioning of already commissioned services has to be dynamic as well as unintrusive. We call this the *challenge of Unintrusive Differentiation of Service Offerings* in a services marketplace [12].

- Business process experts and domain experts along with IT architects and developers play a crucial role in "opening-up" of the platform. Currently, webservice assets are described using a multitude of verbose and formal XML [13] documents. In our opinion, business experts would find it extremely difficult to use these XML specification documents. Instead, they would prefer visual paradigms to support handling webservices. We call this the *Lack of Visual Syntax Problem*.

Service-oriented computing is an evolving and a 'moving-target' discipline. There are several other challenges in the areas of semantics, (dynamic) service composition, interoperability, services management and performance [14, 15]. However, this thesis is primarily motivated from the aforementioned challenges. Hence, the other issues are out of our addressable scope.

1.2 Succinct Research Questions

The questions that our research and this thesis attempt to answer are:

1. How could we support platform owners in methodically "opening-up" their web-business platforms using webservices?
2. How could we represent service artifacts and metadata, in order to increase the longevity of the service-oriented solutions by insulating them from rapid technology evolution?
3. How could we support constant and unintrusive differentiation of commissioned services to keep them competitive in the services marketplace?

1.3 Criteria for Solution

We believe any solution for these research problems would address the following criteria.

Criterion #1: The services must be represented at a conceptual and technology-agnostic level. In order to insulate our service-oriented solution from technology changes, the solution must be captured at an abstract and conceptual-level, agnostic to technology considerations during early-stage development. The service representation must describe both the capability-on-offer – the underlying business functionality – and the terms of offer of the service.

Criterion #2: The high-level conceptual service representation must be easily convertible to executable service specifications. It must be possible to easily convert high-level conceptual service representations to executable service specifications, based on technical considerations like protocol and channel of access.

Criterion #3: The service representation method must have minimal concepts supporting maximal expressiveness. By having minimal representation concepts with maximal expressiveness, business experts would find it easy to use the service representation method to describe various facets of services.

Criterion #4: The service representation should be used by different roles involved during early-stage services development. The service representation must provide different views or perspectives for different roles to describe service artifacts during early-stage services development.

Criterion #5: The service representation must have strong underpinnings in the application domain. The service representation must have underpinnings in the application domain in order to support easy evolution of the solution and provide a common communication lingo between domain experts and the IT experts.

Criterion #6: The service representation must be open-standards compliant and must leverage existing skill-sets and tools. Our service representation method has to be based on open-standards and must leverage existing skill-sets in projects and popular tooling environments.

Criterion #7: The solution must support unintrusive changes to the commissioned services to support competitive differentiation. The solution must support unintrusive changes to the already deployed (commissioned) services in order to differentiate service offerings from that of the competition in the services marketplace.

1.4 Our Proposed Approach

Our proposed approach to finding a solution to the aforementioned challenges is based on OMG's Model-Driven Architecture (MDA) [16] prescription. MDA proposes that the solution be captured using high-level computation-independent models (CIM) which could later undergo a series of transformations to platform-independent models (PIM), platform-specific models (PSM) and finally run-time artifacts [17, 18]. We use MDA recommendations to create service representation using models which helps to capture various facets of services during early-stages of services development. Our metamodels are the cornerstone to our modeling approach, to support platform owners in exposing their business functions as services. Our MDA approach helps us to forward engineer our solution from abstract models to executable webservices specifications.

By representing the solution using high-level models instead of evolving webservice specifications (WS-*), we hope to address the *Evolving Standards Problem*. By capturing the solution space using models, independent of the representational depth and capabilities provided by the current specifications, we hope to completely capture services metadata, thereby addressing the *Lean Services Metadata Problem*. As models are first-class citizens in MDA, we hope that business experts benefit from existing visual modeling tools making our approach business experts-friendly. Therefore we hope to address the *Lack of Visual Syntax Problem*.

1.5 Scope

The focus of this thesis is to understand the different facets of service-oriented development in the context of web business platforms. Using this understanding, we create methodology, modeling perspectives and metamodels to support early-stage services development for platform-owners. We address services granularity [19] by supporting both fine-grained and coarse-grained services. We also address the issue of competitive and unintrusive service differentiation. Throughout this thesis, we have a service provider perspective as our goal is to support the platform-owners. Following are the concrete outcomes of our thesis:

1.5.1 Model Views

We provide modeling perspectives for different roles involved during the early-stage services development. We present these modeling perspectives as *model views*. These model views assist in defining services at different granularity, representing service capability, defining policies associated with services, service realization (or service provisioning), service mediation and service deployment.

1.5.2 Metamodels

Our MOF2-compliant [20] metamodels – Services Metamodel and Resources Metamodel support modeling of both fine-grained and coarse-grained services from the perspective of web-business platforms. These metamodels are open-standards compliant; therefore, existing tools and skill-sets could be deployed to support modeling of services. Even standard transformation languages (like MOF2-QVT [21] and MOF2-Model2Text [22]) could be used to transform the service-oriented solution captured using our metamodels to standard webservice specifications.

1.5.3 Service Flavors Strategy

We provide a competitive and unintrusive service differentiation strategy called the service flavors strategy. Using this strategy, we could isolate the terms of offer of the service from the capability on-offer and competitively alter the terms to differentiate service offerings from that of competition in a services marketplace.

1.6 Outline of the Thesis

In chapter 2, we present the necessary background and context to present our thesis. We discuss the emergence of web-business platforms and the "opening-up" of these platforms through application programming interfaces (web APIs). We also discuss service-oriented paradigm for building loosely-coupled applications and how webservices are the best proposition for web APIs. We discuss in detail the heterogeneities in webservice ecosystems and how model-driven development addresses these heterogeneities by raising the level-of abstraction. We also present a 4-layer SOA architecture used throughout this thesis. Finally, we discuss service granularity.

In chapter 3, we address fine-grained services by modeling web resources. We use the principles of domain-driven design methodology to create a domain model, which forms the basis of our web resources model. The fine-grained services provide basic CRUD

operations on these resources. In chapter 4, we address modeling of coarse-grained services using six model views and our Services Metamodel. In chapter 5, we address modeling of service policies for these services using our service policy metamodel. We explain in detail our domain-independent policy development approach.

In chapter 6, we address unintrusive differentiation of services using our service flavors strategy. Chapter 7 provides a service consumption API model to support creation and evolution of client-libraries to address heterogeneities in service consumption environments. In chapter 8, we present an evaluation of our approach, establish conformance with reference architectures and explain the pragmatics of using our approach. Finally we present conclusions and future work.

1.7 Contribution to Research

The primary contribution of our research and this thesis is our *model views*, our *standards-compliant metamodels* to model service-oriented solutions for web-business platforms and the *service flavors strategy* to differentiate services in a service marketplace.

1.7.1 Vita – Publications resulting from this thesis

Journal publications resulting from this research is as follows:

- Harshavardhan Jegadeesan, Sundar Balasubramaniam: "An MOF2-based Services Metamodel", in Journal of Object Technology, vol. 7, no. 8, Nov-Dec 2008 (to appear)
- Harshavardhan Jegadeesan, Sundar Balasubramaniam: "A Model-Driven Approach to Service Policies ", in Journal of Object Technology, vol. 8, no. 3, Mar-Apr 2009 (to appear)

Conference papers resulting from this research is as follows:

- Harshavardhan Jegadeesan, Sundar Balasubramaniam: "Differentiating Commoditized Services in a Services Marketplace ", in the 2008 IEEE Conference on Services Computing (SCC 2008), Honolulu, Hawaii, USA, July 8 – 11, 2008.
- Sundar Balasubramaniam, Harshavardhan Jegadeesan: "eThens - A Modular Framework for e-Governance", Proceedings of the International Conference on Politics and Information Systems, Technologies and Applications (PISTA 2004),Orlando, Florida, USA, July 2004.

The following paper has been communicated to a journal:

- Harshavardhan Jegadeesan, Sundar Balasubramaniam: “Service Flavors: Differentiating Service Offerings in a Services Marketplace”, communicated to the Journal of Webservices Research on January 22, 2008.

1.8 Summary

Web-business platform owners are “opening-up” their platforms – providing their business functions to be externally and programmatically accessible by the community for co-innovation. They are using the popular open-standard based webservices to expose their business functions by means of web APIs. While doing so, the platform owners are countered by challenges such as the evolving standards problem, lean service metadata problem, lack of visual syntax for describing services and the challenge of unintrusively differentiating service offering from that of competition. In order to methodically expose their platforms and improve longevity and competitiveness of their services, they need to counter these problems. The motivation for this thesis is to address these challenges by adopting a model-driven development approach to “opening-up” of SOA-based web business platforms.

Chapter 2

Background

Web-Business Platforms, Webservices-based SOA and Model-Driven Development

In this chapter, we provide the prelude and necessary context required to present our research. We present details on the emergence of web-business platforms, their technical elements and how they use webservices – a technology manifestation of service-oriented architectures – to expose their business functions to the community. We also highlight the problems and challenges faced by platform owners while using these technologies to expose their business functions. In addition, we provide a brief overview of model-driven development – the approach we think is best suited to tackle these challenges.

2.1 Emergence of the Web-Business Platforms

The world-wide web (WWW) has continuously evolved at a rapid pace from the time it came to existence in the early nineties to the present day web 2.0 [23]. From being a static universe of network-accessible information (read-only web), it has transformed to its present form – dynamic, transaction-oriented and collaborative (read-write web) [24, 25]. Web sites that provided access to own content and services are morphing to support user-generated content and community created value-added services. From being mere websites, they are transforming into 'web platforms'. For example, eBay® which started as an auctioning website has become a complete e-commerce web-business platform creating an entire ecosystem of buyers, sellers and affiliates doing business on the web [1].

2.1.1 "Opening-up" the Web-Business Platform

A software platform [26] is a piece of foundation software around which systems and applications could be built. It is a software program which makes services available to other software programs through APIs. These well-defined interfaces abstract underlying complexity and provide access to platform functionalities. Technically, *APIs open-up the platform for developers to build innovative applications around it*. The operating system is a prime example of a software platform – it provides high-level interfaces to handle hardware resources such as processor, memory and storage. Similarly, by "opening-up" their web-business platforms to the community – developers, business partners and customers – through well-defined interfaces, platform owners foster innovation, emergence of new applications and usage scenarios that they themselves might not have envisaged.

Analogous to the operating systems, web-business platforms expose their business functions through well-defined interfaces or web APIs to the community. "Opening-up" of web platforms through web APIs has created new possibilities [27]. Firstly, custom functionality can be built by customers based on unique business needs without using website user-interfaces. For example, with eBay® APIs it is possible to list items for auctioning on the eBay® marketplace, without using their web site. A manufacturing company can auction excess inventory in the eBay® marketplace directly from its enterprise-resource planning system using APIs. Secondly, an exciting genre of web application hybrids, commonly known as mash-ups [28] can now be built using web APIs by leveraging content from more than one source of information (content provider). Thirdly, entrepreneurs and developers can exploit synergies between different content and service providers to build innovative end-to-end applications for customers.

2.1.2 Web Application Programming Interfaces

In the previous section, we discussed that web APIs are used to open-up web business platforms. The most important technical criteria for web APIs is that, they must be accessible by heterogeneous consumers, especially, from a variety of technology platforms such as .NET® [29], Java™ [30], open-source platforms (such as PHP, Pearl and Python), propriety platforms (such as SAP®) and browser environments (JavaScript™). As

webservices [31] are based on open-standards and are widely adopted, they meet the technical criteria of platform-independence. Therefore, webservices emerge as a natural choice for web APIs. Presently, apart from web APIs, the platform owners also provide language-specific application libraries (especially .NET, Java and PHP language libraries) to be used directly in client code. These libraries are webservice client-proxy programs that abstract message-based interactions with the remote web service. We address client-libraries in chapter 7.

2.2 Service-Oriented Architectures and Webservices

Service-oriented computing paradigm deals with organizing and utilizing distributed capabilities under the control of different ownership domains [32]. Every service represents a capability on-offer – a business function – and the terms at which this capability is offered (terms of offer) [33]. The capability on-offer satisfies the goal of the service consumer under the constraints of the terms of offer. We refer to the OASIS SOA Reference Model (herein SOA-RM) [32] for a formal and broad definition of service – *A service represents an underlying capability offered by a service provider that meets the goals of one or more service consumers.* Service-Oriented Architecture (SOA) considers services as first-class entities to build applications [34]. *Webservices – a technology implementation of SOA – are self-describing, self-contained components that can be automatically discovered and invoked using open-standards*. Fundamentally, SOA is an architectural style while webservices are practical implementations based on the SOA architectural style. Webservices are popular due to the fact that they are based on interoperable open-standards such as SOAP, WSDL which make them platform-independent.

The term 'web service' has been commonly used while referring to SOAP based webservices. However for our research, *the term 'webservice' encompasses all services offered on the web (web-based services), based on open-standards, accessible in a loosely-coupled fashion through message-based interactions*[1]. Apart from the standard SOAP-

[1] Note the use of "webservice" instead of "web service". We use this subtle difference to refer to our definition of webservices.

based services, this could include services based on the REST (**RE**presentational **S**tate **T**ransfer) architectural style and Plain Old XML (POX) services. The SOAP-based services could still further be document-style or RPC-style supported through a variety of transport protocols, the most popular being HTTP [35, 36]. The REST-based services support message exchange over plain-HTTP in either XML or in JSON (java script object notation) [37] payloads. REST-style services could also be based on popular syndication protocols such as ATOM and RSS [38].

2.2.1 Service Granularity

Granularity is a relative measure of how broad the level of interaction between a service consumer and a service provider has to be. Service granularity refers to scope of business functionality a service exposes, thus addressing the level of encapsulation of a particular business capability to support the "loosely-coupled" philosophy of service-oriented architecture [39]. Right service granularity is critical to achieve service reusability. The approach we take to build services has an impact on granularity. Services could be built using a "code-first" or an inside-out approach. In the inside-out approach existing IT assets are exposed by using webservices interfaces. Alternatively, the "contract-first" or the outside-in approach is purely driven by stakeholder requirements. By using the latter approach, services could be built in the right granularity [40].

Broadly, services could be either fine-grained or coarse-grained [41]. Fine-grained services address a small unit of functionality. In contrast, coarse-grained services address a larger functionality. In our thesis, we view fine-grained services as services providing CRUD (Create, Retrieve, Update, Delete) [42, 43] manipulation of web resources [44]. Fine-grained services are primarily used in user-interfaces and mash-ups. They can also be used in a few application-to-application integration scenarios. We address fine-grained services in chapter 3. Coarse-grained services, on the other hand, handle larger business tasks. They support triggering of business functions in the platform, notifications of events and functions that require manipulation of one or more resources. They could be used in business-to-business integration scenarios and some application integration scenarios. We address coarse-grained services in chapter 4.

SAP's Enterprise Services [45] defines three types of service interfaces – A2X (application to 'any'), A2A (application to application) and B2B (business to business). The A2X services are fine-granular while the A2A and B2B services are coarse-granular in nature. In addition, by convention, A2X services are always synchronous, while A2A and B2B services could be either invoked synchronously or asynchronously.

2.2.2 Heterogeneity in the webservice ecosystem

Various service delivery styles and protocols have led to heterogeneity in development approaches as well as the webservices ecosystem as a whole [46]. The web services interoperability specifications [47] addresses interoperability of WS-* web services from different vendors through the WS-I Basic Profile [48]. At one end, the heterogeneity in the ecosystem is due to evolving standards, at the other, it is due to different interpretations of standards by implementing vendors. Nevertheless, it is important for the platform-owners to support various styles and protocols to increase platform usage and adoption among the community. Recently, there is a growing understanding that each of these service delivery styles and protocols are suited for a specific-purpose or an environment. For example, XML payloads are better suited for webservices consumed by third-party applications, while JSON payloads are easier to handle in browser-based environments, due to their native JavaScript support. SOAP-based services are useful for coarser service-cuts, while REST is best suited for fine-granular services – primarily CRUD services on web resources [49, 50]. We discuss this further in chapter 3.

Thumb rules over appropriateness of service delivery styles and protocols are beginning to emerge. However, platform-owners must have the flexibility to offer their business functions as services in whichever way they deem fit. It should be possible to offer the same service in different delivery styles or protocols to support different consumer's technical environments. Presently, web business platforms such as eBay®, Amazon®, Google® and many others have been providing web APIs which are based on both SOAP-based and REST-based service interfaces [51].

2.2.3 Reference Architectures for SOA

Though SOA offers significant advantages by fostering loosely-coupled applications, creating a SOA-based solution is rather difficult. Specifically, it is difficult to systematically organize and implement the solution [52]. In order to create SOA based solutions easily and in an organized fashion, reference architectures and patterns have emerged [53-55]. The Service-oriented Solution Stack (S3) [56], a popular vendor-neutral SOA reference architecture provides a metamodel and a flexible nine-layer solution stack for SOA solutions based on best industry practices. Each layer addresses specific logical and physical perspectives, thus helping in effective separation of concerns. The S3 can be employed with methods such as IBM's Service-oriented Modeling and Architecture (SOMA) [52] to create SOA-based solutions. For the purpose of this thesis, however, we use a slightly altered and simplified "4-layer architecture" to present our work (fig. 2.1). Even though our discussions are based on our layered architecture, the methods we present are independent of it and can be applied in the context of any reference architecture. We briefly explain our 4-layered SOA architecture below:

- **Operational Systems Layer:** The operational systems layer consists of home-grown custom applications, legacy systems, packaged enterprise systems and databases. The operational systems layer depicts the existing IT-assets in the system landscape.

- **Domain Layer:** We use a domain layer in a broader context than that of the service components layer in S3. The domain layer contains domain entities and business functions and can be organized using a Business Reference Model (BRM). The BRM in the Federal Enterprise Architecture (FEA) [57] could be used for this purpose; however, this is not in the scope of our solution. The BRM hierarchically organizes the business functions in a given domain. These business functions are ideally exposed as services to the community. The domain layer also attempts to bridge the gap between business and IT. The domain experts are closely involved in defining the business entities and the related business functions in the domain layer, as well as organizing them based on the business reference model.

- **Services Layer:** The services layer consists of all services defined in the solution. These services are offered by the platform owners based on different service delivery styles and protocols. The services could be of different granularities ranging from a coarse-grained service used for business-to-business communication or a fine-granular CRUD service consumed in mashup interfaces.

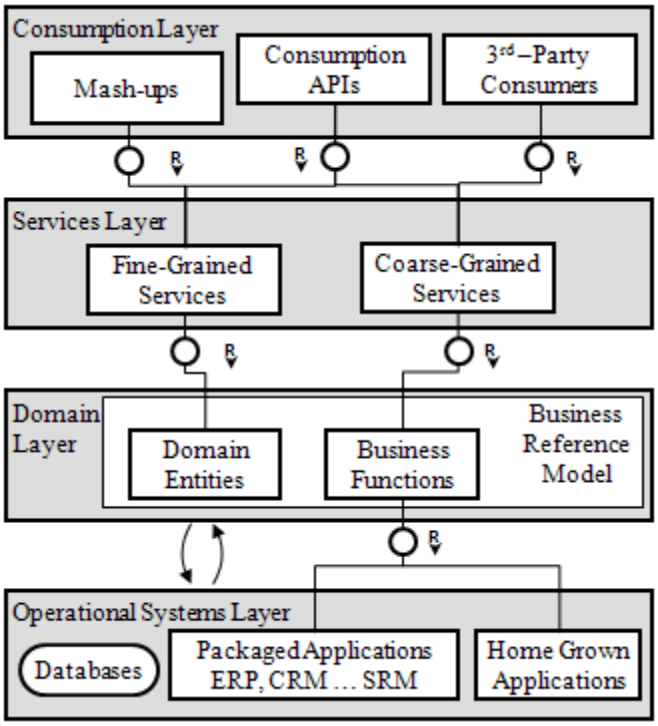


Fig 2.1: Simplified 4-Layer SOA Reference Architecture

- **Consumption Layer:** The consumption layer consists of applications, business processes, mash-ups and other 3rd party applications in which services are consumed. It also contains service consumption APIs, popularly known as client-libraries or consumer proxies.

2.3 MDA - Raising the Level of Abstraction

In order to address heterogeneity in the webservices ecosystem, we adopt a model-driven approach prescribed by Model-Driven Architecture (MDA) to services design and development. Before understanding MDA, it is essential to have a common definition of a model. *A model is an abstract definition of (part of) a system written in a well-defined language* [58]. A well-defined language is a language with well-defined form (syntax), and meaning (semantics). OMG's MDA [59] is a framework for software development which prescribes using models as first-class entities for specifying a software system and its

functionality. MDA effectively separates system specification from its implementation, keeping the focus on abstract and conceptual system definition in early-stages of development. In addition, it helps postponing technology decisions to appropriate later stages of the development cycle.

MDA supports separation of concerns by using three-layers of models – the Computation Independent Model (CIM), the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM) [60]. The CIM model elements capture the problem domain from a functional and a business viewpoint. The PIM model elements build on the CIM and add computation specific aspects without being concerned with the implementation platform. Finally the PSM builds on the PIM to add technology platform specific implementation details. A PSM can then be transformed into executable code and specifications (fig. 2.2).

A series of transformations (a.k.a. model transformations) is used to convert from CIM to PSM. Behind the model specification and transformations, lie a set of metamodels – fundamentally a model is an instance of its metamodel. *The metamodel is a language with well-defined formal syntax and semantics to describe models* [16, 20]. The metamodel is essentially a domain-specific language (DSL) [61] to define systems in a particular domain. OMG's Meta-Object Facility (MOF2) provides the abstract syntax to define the modeling constructs of a metamodel. MOF2 is tightly aligned with the UML2 Infrastructure [62] – MOF2 and UML share a common and unifying set of modeling elements called the *kernel*. Transformation languages such as MOF2 Queries/ Views/ Transformations (MOF QVT) and MOF Model2Text support model to model and model to text transformations respectively. UML in its new avatar – UML2 – is a family of languages [63] supporting the creation of domain-specific languages to address modeling of specific domains. This is possible because of the structured 4-layer UML2 hierarchy [64] (fig. 2.3). The M3, meta-meta model layer has basic model elements to support creation of a metamodel. The M2 layer represents the metamodels such as UML – to support modeling of object-oriented systems, Common Warehouse Metamodel (CWM) [65] – to support modeling of databases and warehouses and Ontology Definition Metamodel (ODM) [66] – to support modeling of ontologies.

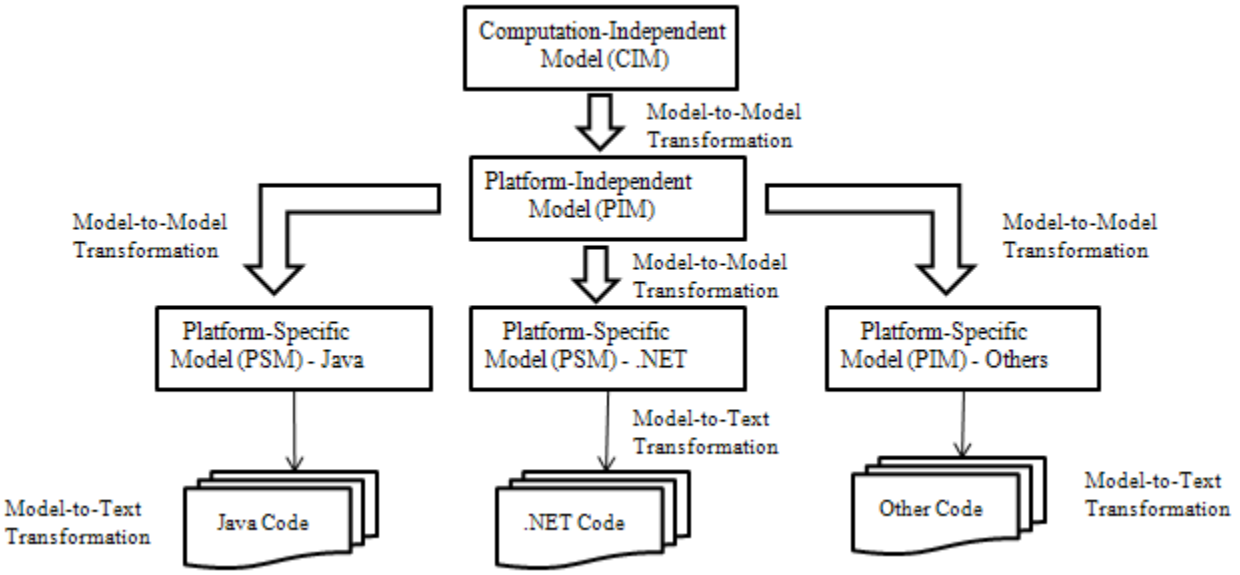


Fig 2.2: The MDA Layers and Transformations

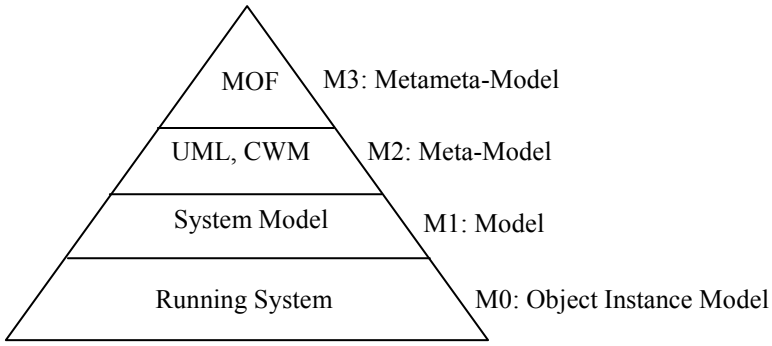


Fig 2.3: UML 4-layer Hierarchy to Support Family of Languages

The M1 layer contains the system model instantiated by the metamodel and the Mo layer contains the object instance model representing the running system. The prevalent approach to create a domain-specific modeling language is to create a MOF2-based metamodel with domain-specific concepts as model elements along with a corresponding UML2 Profile [67]. The UML2 profile helps to leverage existing tools and skills. *A UML2 profile is a stereotyped package that contains model elements that are customized for a*

specific purpose using extension mechanisms such as stereotypes, tagged values and constraints. We also adhere to this approach by creating metamodels and corresponding UML2 profiles to model services.

2.3.1 Model Transformations

Model Transformations are central to model-driven development. *A transformation is an automatic generation of a target model(s) from a source model(s), according to a transformation definition* [68]. A transformation definition is a set of transformation rules that describe how a model in the source language can be transformed into one or more constructs in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed to one or more constructs in the target language [69].

Model Transformations could either be model-to-model transformations or model-to-text transformations, supporting transformations of a model to another model (e.g. PIM to PSM) or text (e.g. PSM to code). The vision of model-driven development is to shift the focus from programming to capturing the solution in conceptual models, thereby increasing the longevity of the solution. For this vision to become a reality, it must be possible to transform high-level conceptual models to useful executable specifications or code. Using one or more input models and producing one or more output models, requires a good understanding of the formal abstract syntax and semantics of the input and output models. Numerous model-to-model transformation languages are emerging. Some of them are ATL (ATLAS transformation language) [70], transformation using XSLT [71] on XMI [72, 73] representations, and Kermeta [74]. Again, these are based on different approaches such as direct manipulation, graph-transformation and other hybrid approaches [75]. In order to standardize the model-to-model transformations, OMG has come out with MOF2 QVT (Query/View/Transform). A classification of model transformation approaches is presented in [75]. There are also graphical model transformation languages such as MOLA (Model transformation language) [76], UMLX [77] and GReat [78] which provide a visual syntax to support model transformations.

Model-to-text transformations are also becoming popular. Models can be transformed to executable specifications, code and other XML-based artifacts (deployment descriptors).

Some of the model-to-text transformation languages include MOFScript, Xpand, and Java Emitter Templates (JET) [79]. These transformation languages are either based on visitor-based approach or a template-based approach, the latter being more popular. The OMG's MOF2 Model to Text (mof2text) standard addresses how to translate a model to various text artifacts using a template-based approach.

2.4 Summary

In this chapter, we presented a background of web-based business platforms. We explained platform strategy – operationalized by exposing a web-business platform using open-standards based webservices. We also presented details on model-driven development – our chosen approach to address challenges faced by platform-owners. We follow the prescription of Model-Driven Architecture (MDA) to support platform owners in systematically "opening-up" their web-business platforms using webservices-based web APIs.

We specify services and web resources precisely using technology-agnostic, high-level conceptual models. These models can later be translated to concrete executable specifications or code using transformations. We address both fine-grained and coarse-grained services (see section 2.2.1). Fine-grained services support CRUD-based access to web resources. Web resources are basically real-world objects in the business domain, captured in the domain model. We use a domain-driven design approach to create an expressive domain model of the application domain (chapter 3). Certain domain entities in the domain model could be exposed as web resources. We have a resource model representing certain domain entities which are opened-up for manipulation. Fine-grained services support fine-granular manipulation of these web resources. For coarse-grained services, we define a MOF2-based Services Metamodel (a M2-level model in the 4-layer UML hierarchy) to support various facets of modeling coarse-grained services (chapter 4). We compliment our Services Metamodel with a UML Profile to leverage existing modeling tools. We also support independent service policies development using a policy model (chapter 5).

In summary, platform-owners face complex challenges while operationalizing their platform strategy. There is plenty of research, though conceived in a different context, which partly address these challenges. However, we strongly believe that they do not provide a comprehensive solution to address web-business platform challenges (as discussed in chapter 1). In subsequent chapters, we explain different components of our solution. We also attempt to provide a comprehensive related works section in each chapter where we compare and contrast our solution components to that of other existing approaches.

Chapter 3

Modeling Web Resources and Fine-Grained Services

Using Domain-Driven Design Techniques to Model Web Resources and Fine-Grained Services

In this chapter, we address modeling of web resources and subsequently the fine-grained services which provide CRUD-manipulation on these resources. *Resource is a real-world entity in the domain that would have an identifier* [80]. *Apart from having an identifier, a resource has a name, has a reasonable representation, a resource description and is owned by a person or an organization* [81]. Representation of a resource reflects the state of the resource. Resource is a fundamental concept that underpins the web. While a service represents an underlying *capability* offered by a service provider, a resource is an underlying entity in a particular domain – a domain entity. In an abstract sense, a service is activity-centric and focused on 'verbs', in contrast to resources which are focused on 'nouns'. A resource represents the state of the domain entity explicitly while a service represents the state implicitly. In this chapter we focus on modeling web resources.

3.1 Heterogeneity Due to Conflict of Styles

In chapter 2, we discussed the heterogeneities in the SOA landscape. These heterogeneities could be due to technical protocols (WS-*), payloads (XML, JSON) or even due to styles (SOAP vs. POX vs. REST) [50]. We also discussed how model-driven development – the approach we take in our thesis – helps in managing these heterogeneities and increasing the longevity of the solution. In this section, we would like to discuss briefly about the heterogeneity due to *conflict of styles* – the REST vs. SOAP-

style debate. Although we acknowledge that this is not significantly relevant, the underlying issues it highlights are more important than the debate itself. The fundamental principle of SOA is loose-coupling. This debate is about which style supports development of loosely-coupled and scalable applications based on service-oriented architectures. Our aim is to provide the necessary abstraction in early-stage design and modeling to support different styles, payloads and protocols. Nevertheless, we still consider it important to discuss these conflicting styles to emphasize the difference between *activity-oriented* vs. *resource-oriented* focus of these styles [82].

Representational **S**tate **T**ransfer (REST) is an architectural style which underpins the web – it supports independent development and therefore scalability in web architectures. The central principal in REST is the existence of *resources*, each of which could be identified by a globally unique identifier – the *Uniform Resource Identifier* (URI) in the context of the web. In addition, each resource has a standard representation which reflects the state of the resource. REST-style is supported by the Hyper-Text Transfer Protocol (HTTP), which provides standardized interfaces to manipulate these resources through the POST, GET, PUT, and DELETE methods [10] (fig 3.1). These methods support CRUD (Create, Retrieve, Update and Delete) operations on the resources. While the HTTP methods are a part of the HTTP-header, the payload consists of a resource representation in either XML or in JSON. In terms of service granularity, REST-based services support fine-grained CRUD services which could be readily consumed in mashup user interfaces or in other web applications.

Before the REST-style was articulated by Roy Fielding in this doctoral thesis, webservices were predominantly based on SOAP and WS-* protocols. A significant point to note is that, Fielding presented REST in the context of uniform information access (*'resource'* focus) rather than remote function calls (*'activity'* focus). While every practitioner has his own preferences and loyalties, it is important to use appropriate styles to solve different classes of problems [50, 49]. Let us contrast the resource-oriented REST-style to that of the activity-oriented SOAP-style using an example. Consider an online LISTINGSERVICE such as the one offered by Shopping.com®[2], which supports pricing comparison across online shops. The LISTINGSERVICE supports the online shops and merchants to list their products in Shopping.com® listings. A buyer could search and compare products in

[2] The services and scenarios we describe in the context of Shopping.com is fictitious, however it closely represents real-world scenarios.

different online shops through Shopping.com®. Once the buyer decides to buy a particular product, he is redirected to the online shop. Fig 3.2(a, b) depicts both a REST-based interface and a SOAP-based interface for the service. Note that while the SOAP-interface is activity-centric (note the *verbs* e.g. CreateNewListing), the REST-interface is resource-centric (note the *nouns* e.g. Listing).

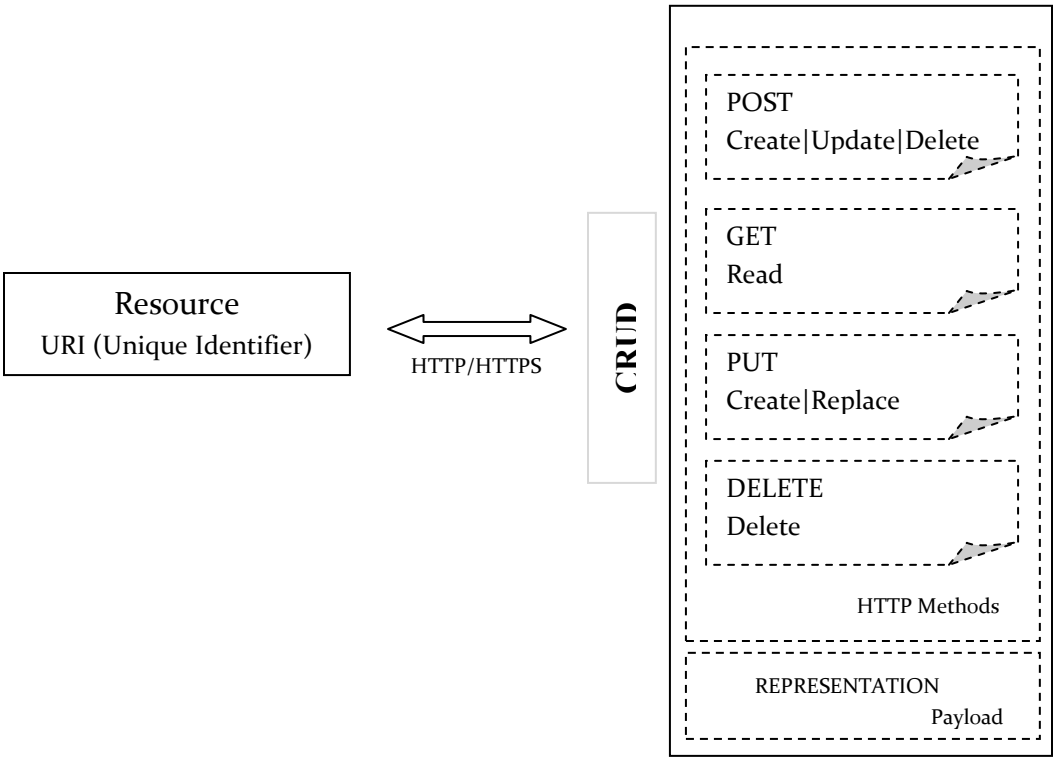


Fig 3.1: REST-based Standardized HTTP Interfaces to a Resource

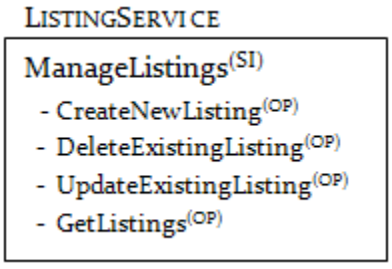


Fig 3.2 (a): SOAP-based Interface to Shopping.com® Listings

Sample Listing - XML representation

```
- <l:Listing xmlns:l="http://shopping.fictitious.com"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <ProductID>00345</ProductID>
    <Name>iPod Classic</Name>
    <Description>Apple's Fifth Generation iPod</Description>
    <Specification
      xlink:href="http://shopping.fictitious.com/listings/aoed-156-
      w2rdf/specification" />
    <UnitCost currency="EUR">210</UnitCost>
    <Quantity>10</Quantity>
  </l:Listing>
```

Create	*HTTP Method*	POST/PUT
	Payload	Listing XML instance
Retrieve	GET: http://shopping.fictitious.com/listings/aoed-156-w2rdf GET: http://shopping.fictitious.com/listings?query=ipod+classic (Lists all iPod Classic listings from different online shops)	
Update	*HTTP Method*	POST/PUT
	Payload	Listing XML instance
Delete	*HTTP Method*	DELETE
	Payload	Listing XML instance

Fig 3.2 (b): REST-based Interface to Shopping.com® Listings

With time some thumb rules have evolved in the community in choosing one approach over the other [50]. In our opinion, the most important of these is *service granularity*. For fine-grained services which support CRUD like operations, REST-style is considered

minimal; however for coarse grained services which represent underlying business capabilities, SOAP-style is preferred. In addition, for direct consumption in mashup user interfaces and web applications, fine-grained REST-style services are preferred. Coarse-grained SOAP-style services are used in application integration or for business-to-business (B2B) scenarios. Apart from service granularity, the second distinction is the *informational vs. transactional nature* of the scenarios in general [83]. Our suitability argument that is for information access (simple 'read/write') scenarios, REST-based services are simple and best suited and for transactional scenarios SOAP-based services are best suited. However for platform-owners, consumer preferences and considerations also play a big role in deciding one style over the other. We would like to emphasize our suitability argument with empirical examples. Let us consider a practical example of Google™ Services. Google exposes its different web-platforms in different ways. Google Base [84] is a platform through which one could post any type of semantic content to Google and make it searchable from other Google properties. A typical scenario would be for merchants and online sellers to post product information or their entire catalog to Google Base and make it accessible through Google Product Search (earlier Froogle®) [85]. The Google Base platform is exposed using REST-based services through the Google Base API. The Google Base API is based on Google Data APIs [86] (*GData* for short), a standard protocol for reading and writing data on Google properties. GData works based on popular syndication protocols such as RSS and Atom Publishing Protocol (APP) [87]. The GData is built in the spirit of the REST approach. A common feature of all Google properties using REST-based GData (like Base, Blogger, Calendar, Contacts etc.) is that they provide *fine-grained* information access. Contrast this with other Google platforms such as Google Checkout [88] and Google Adwords [89]. Google Checkout supports safe and single-login purchases across different online stores for customers and a new and efficient way to process sales for merchants. Google Adwords platform is a targeted advertisement platform. Both Checkout and Adwords API services are *coarse grained and transactional in nature;* therefore, they are SOAP-based. It is also possible for a provider to provide access to its web-platforms based on both the styles, and leave the choice entirely to the consumer. Consider the example of eBay® Shopping and Merchandizing APIs [90], they provide both SOAP and REST-based interfaces with support for different payloads and protocols.

It is important for web business-platform owners to support both SOAP-based as well as REST-based interfaces to their platforms. This will facilitate wider platform adoption, thereby supporting diverse service consumption and platform usage scenarios. Hence our

approach should focus on modeling resources and fine-grained access to these web resources, abstracting protocol and style details.

3.2 The Domain-Driven Design Approach

Irrespective of the styles and protocols, there is a fundamental need to model web resources and fine-grained services. Web resources are basically domain entities in an application domain. For example, a *Product Listing* is a domain entity in the e-commerce domain. We would need fine-grained CRUD services to manipulate product listings. There is a need to identify domain entities in a particular domain, which would eventually be resources. Once resources are identified, they have to be adequately represented. In order to have a business domain focus in our solution, we follow a domain-driven design (DDD) approach [91] – an extensive use of domain models to identify and model domain entities and eventually resources.

Domain modeling is an activity that would happen in the domain-layer of our simplified 4-layered SOA architecture. The DDD approach, apart from supporting identification of resources through domain entities, also supports realization of fine-grained services associated with these resources. So what is domain-driven design? The goal of domain-driven design is to put the domain model at the heart of developing software, keeping the focus on models rather than technology. *A domain model is an abstract representation of a particular domain crystallized by a domain expert* [92]. The domain model provides the structural underpinnings for our solution and is focused at describing the domain-layer.

3.2.1 Modeling the Domain using Domain-Driven Design Approach

As a first step, we use the principles of domain-driven approach to model the application domain. The domain entities in the domain model would eventually be web resources. We model the domain based on the tenets of domain-driven design. In order to support the use of domain-driven design, we create our *DDD Metamodel, a domain-specific language for domain-driven design where the model elements are extensively borrowed from*

the domain-driven design approach. Our DDD Metamodel is a MOF2-based M2-layer metamodel compatible with UML family of languages. It extends the *UML Infrastructure Library::Core* package (fig 3.3). The DDD Metamodel (fig 3.4) supports our **Domain Model View**, a model view at the domain layer. In order to leverage existing UML tools to model the domain, we have also created an UML2 Profile (fig 3.5). The domain model created using the DDD Metamodel could be transformed to code in different provisioning languages to support implementation, using model-to-text transformations.

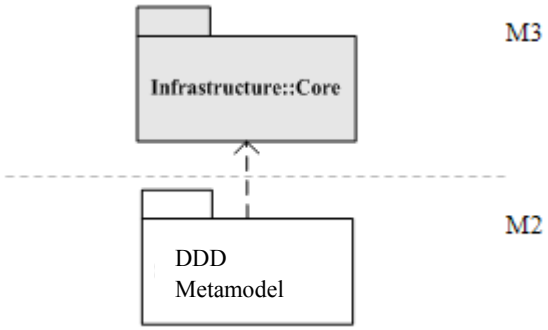


Fig 3.3: The DDD Metamodel (a M2-layer metamodel)

Key Classes and Associations

Entity: Entity is a domain object which is defined primarily by its identity [91]. Entity is something that has continuity through its life-cycle and is distinguished by its identity rather than its attributes. An entity could be independent or be part of an Aggregate (discussed below). It could also be an aggregate root. The attribute *isAggregateMember: Boolean* determines if the entity is part of an aggregate or not. An entity has a *globalIdentifier: UUID,* natural identifiers from the domain (e.g. social security number for an employee) and a lifecycle state. The Identifier, UUID and the LifeCycleState extend the *Core: Property*. An entity is a *RepositoryItem*, and a Repository (discussed below) manages the life cycle of an entity. In addition, an entity is created by a Factory (discussed below). An entity could conform to one or more Specification (discussed below). An entity has an operation *checkConsistency* which ensures that the entity is consistent and conformant with specifications. It extends the *Core: Class*.

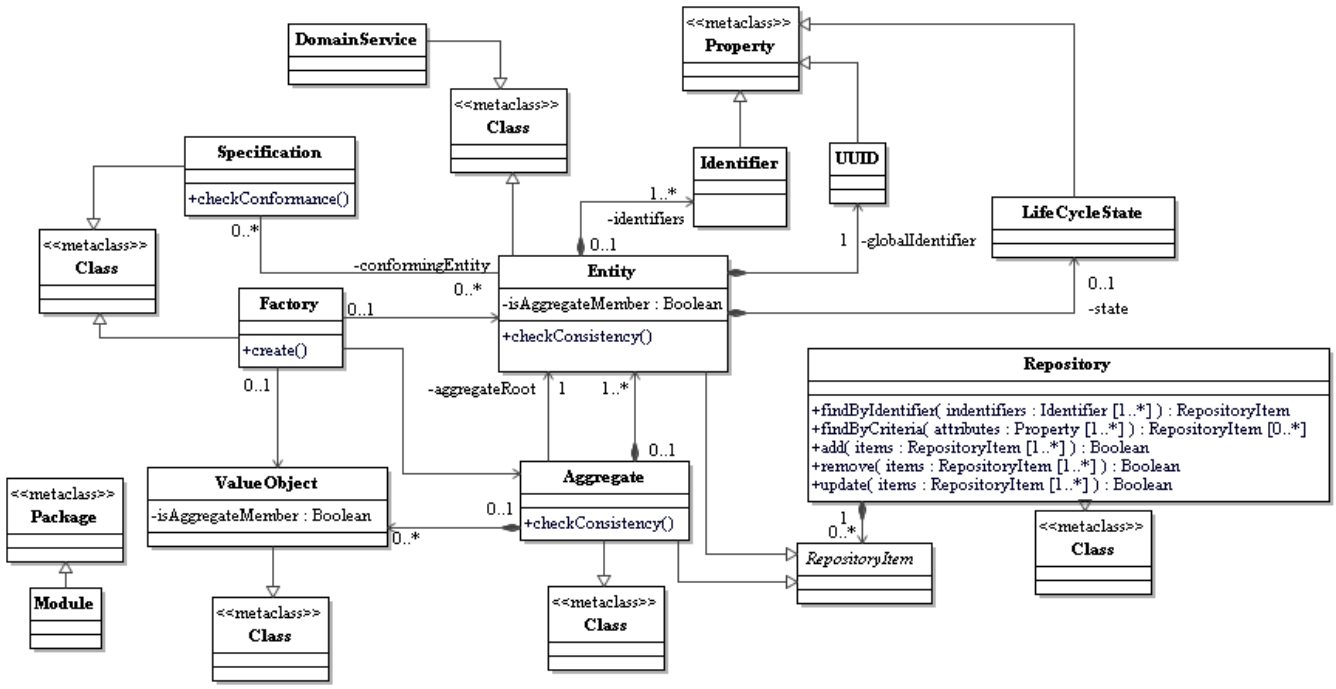


Fig 3.4: The DDD Metamodel – Domain Model View

Value Object: Value object is an object in the domain which has a descriptive nature but with no conceptual (or natural) identity in the domain [91]. A value object can also be an aggregate member (*isAggregateMember: Boolean*). Value objects are immutable and can be created by a Factory. It extends the *Core: Class.*

Aggregate: An aggregate is a cluster of associated objects (entities, value objects) which we treat as a single *unit* for the purpose of data changes and maintaining consistency [91]. Every aggregate has a root entity which is the only object that could be accessed directly outside the aggregate boundary. An aggregate is created by a Factory. An aggregate is also a *RepositoryItem.* An aggregate also has a *checkConsistency* operation which ensures the consistency of the aggregate. It extends *Core: Class*.

Module: A module logically partitions the domain. The partitioning is from a conceptual and domain perspective [91]. It extends *Core: Package.*

Factory: A factory provides the necessary encapsulation to create entities, value objects or aggregates [91]. Factories do not necessarily represent a domain concept, but are responsible for constructing one. It has a *create* operation which is responsible for creation of entities, value objects or repositories. It extends the *Core: Class*.

Repository: A repository is a conceptual set of all objects of a certain type [91]. It acts as a collection with elaborate querying capabilities (finding objects by their identifiers (if any), or by other attribute criteria). A repository supports addition of new objects as well as updating and deleting existing objects. It supports operations such as *add, remove, and update* as well as querying operations such as *findByIdentifier* and *findByCriteria*.

Specification: A specification provides a concise way to capture business rules [91]. Normally such rules would be hard-coded in business logic. Specification makes it easy to specify rules explicitly in the model. A specification has a *checkConformance* operation which checks if an entity conforms to the specification. A specification could have one or more conforming entities. It extends the *Core: Class*.

Domain Service[3]: There are some domain operations which do not fit naturally in an entity or a value object. A domain service is an activity in the domain rather than an entity that represents these domain operations [91]. At the domain layer, the means of providing access to a domain service (through distributed architectures such as RMI, CORBA or SOAP [93]) is not as important as the design decision to carve off a particular system responsibility as a service itself.

[3] In Domain-Driven design the Domain Service is actually called a SERVICE. We call it domain service just to differentiate it from the services in the services layer

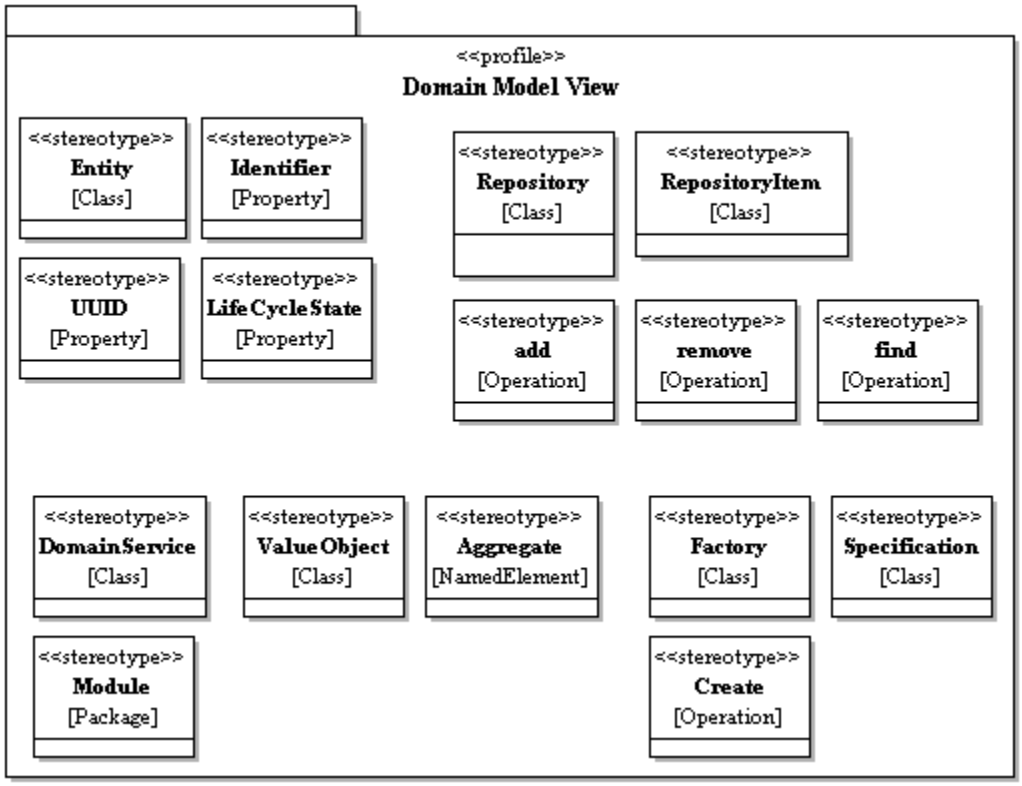


Fig 3.5: The Domain Model View (UML2 Profile)

3.2.2 Resources Model

Using the principles of domain-driven design and with the help of our DDD Metamodel, it is possible to model the application domain in the domain-layer. Once we have the domain model and the domain entities, *web resources* have to be identified and created. As resources have identifiers, the model entities from the DDD Metamodel – Entities and Aggregates – would be ideal candidates for web resources. However it is possible that based on business requirements of the web business platform, only certain Entities and Aggregates from the domain model would be exposed as resources. We have a resources model that is derived from the domain model through a model-to-model transformation (fig 3.6.A) to represent resources.

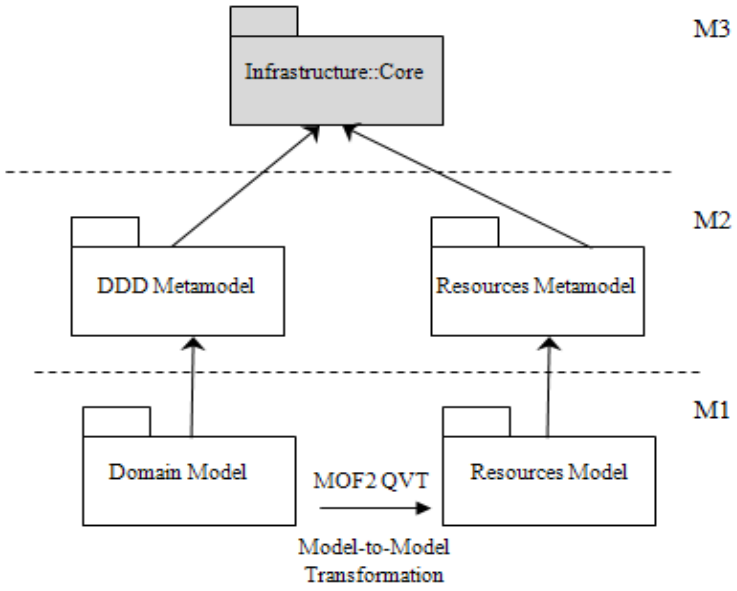


Fig 3.6 (A): Model-to-Model Transformation (Domain Model to Resources Model)

Fig 3.6 (B) provides an overview of the model-to-model transformation using UMLX visual syntax. The resources model is based on our Resources Metamodel (fig 3.7).

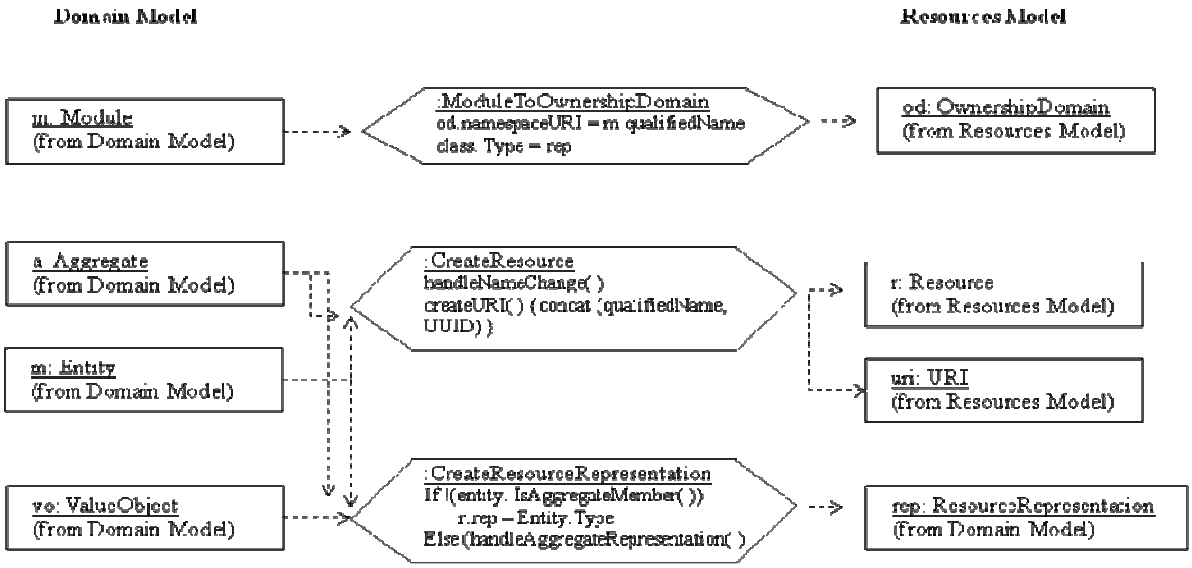


Fig 3.6 (B): Model-to-Model Transformation – UMLX Visual syntax

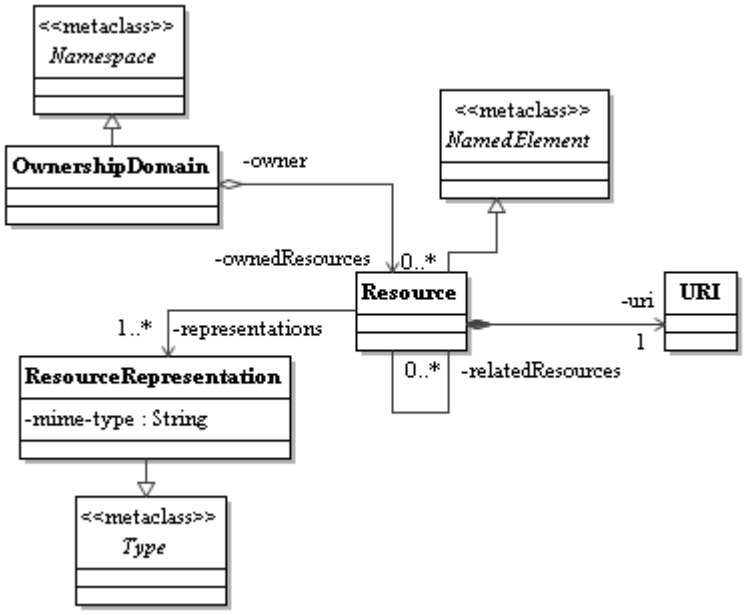


Fig 3.7: Resources Metamodel

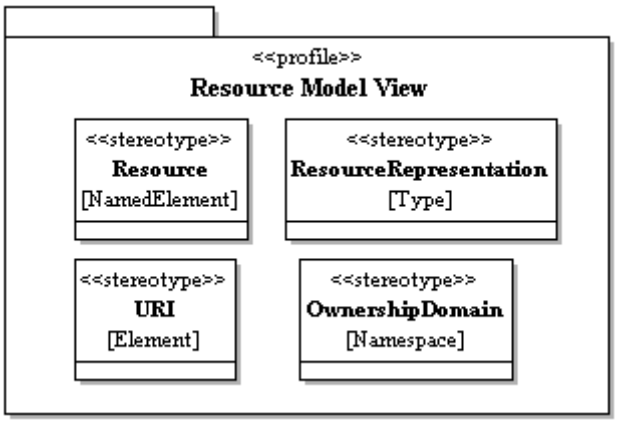


Fig 3.8: Resources Model View

Our Resources Metamodel is also a MOF2-based M2-layer metamodel which extends the *UML2 Infrastructure::Core*. We also have an associated UML2 profile (fig 3.8)

Key Classes and Associations

Resource: A resource is an entity in the domain with an identifier [94]. A resource could have related resources and has an identifier URI. It also has one or more resource representations. An ownership domain owns resources. It extends *Core: NamedElement.*

URI: An URI uniquely identifies a resource. It is derived from the qualified name of an entity as well as the entity identifier (UUID) in the domain model.

Resource Representation: A resource representation provides a semantic representation for a resource through a data schema. A resource could have more than one representation. The representation could be based on XML, JSON, RDF or any other description language identified through the *mime-type* attribute. It extends the *Core: Type.*

3.2.3 Uniform Access to Resources

Once the resources are identified, it is essential to support uniform access to manipulate resources. We support the standard fine-grained CRUD pattern to access to resources in the services-layer irrespective of whether the platform owner decides to expose the resource through a SOAP-based interface or through REST. Creating REST-style interfaces for manipulating a resource is straightforward as it fits with the CRUD pattern. If the platform owner chooses to expose the resource through a SOAP-style service for whatever reason, then a standard service is created automatically at the service-layer to manipulate the resource. We call it the MANAGE<RESOURCE>SERVICE. The manage service for a resource would support operations such as creation (Create<Resource> $^{(OP)}$), change (Change<Resource> $^{(OP)}$), delete (Delete<Resource> $^{(OP)}$) and querying a particular resource (fig 3.9). The query on the resource could either be based on resource identifier (Query<Resource>ById $^{(OP)}$) or through some criteria (Query<Resource>ByCriteria $^{(OP)}$).

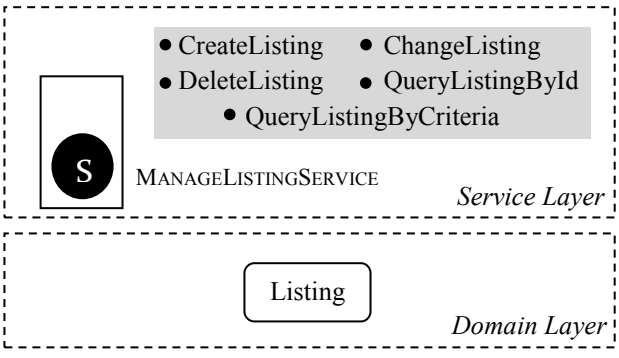


Fig 3.9: MANAGELISTINGSERVICE – *SOAP-based Service*

The manage service at the service-layer is a fine-grained service to manage and manipulate resources through a uniform mechanism. Whichever style the resource is exposed, the provisioning (implementation) for the standard CRUD pattern to manipulate the resource is supported in the domain-layer. This is supported by the operations of the *Repository* of the corresponding entity or the aggregate. The standard Repository operations such as *add, remove, update, findByIdentifier and findByCriteria* support the uniform CRUD pattern interface.

3.3 Model to Executable Specifications

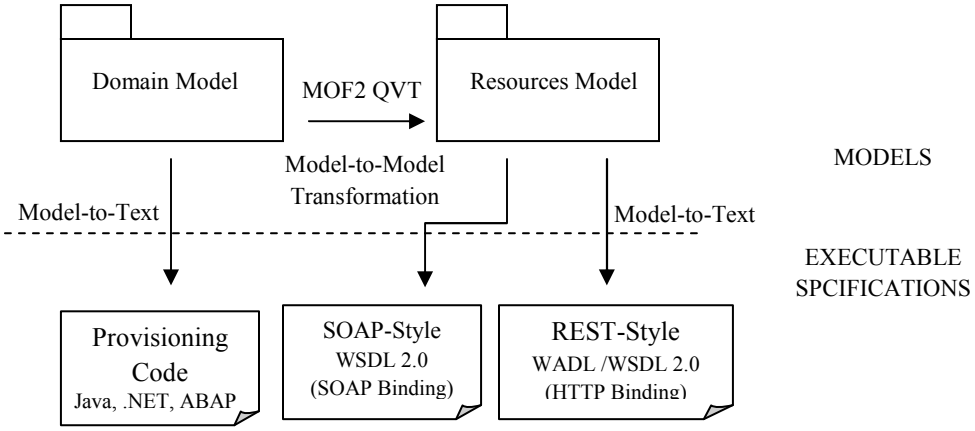


Fig 3.10: Transformation to Executable Specifications

Fig 3.10 explains the transformations from the models to executable machine process-able artifacts. The domain model is transformed to provisioning (implementation) skeleton code using a model-to-text transformation. The skeleton code could be in Java, .NET or any other provisioning language using different model-to-text transformations. Earlier, we discussed that the domain model is converted to resource model using a model-to-model transformation. The resource model has to be converted to executable specifications. The choice of specifications is based on if the resource is exposed through a SOAP-style interface or through a REST-style interface. For the SOAP-style interface we transform the resource model to WSDL 2.0 with SOAP-bindings. For the REST-style interface the resource model is transformed to either WADL [95] (Web Application Description Language) or to WSDL 2.0 with HTTP-bindings. However, it is possible to

transform the resource model to any other machine process-able description such as APP (Atom Publishing Protocol) etc. This is achieved by creating new transformations for REST-style services.

3.4 Modeling an Online Shopping Scenario

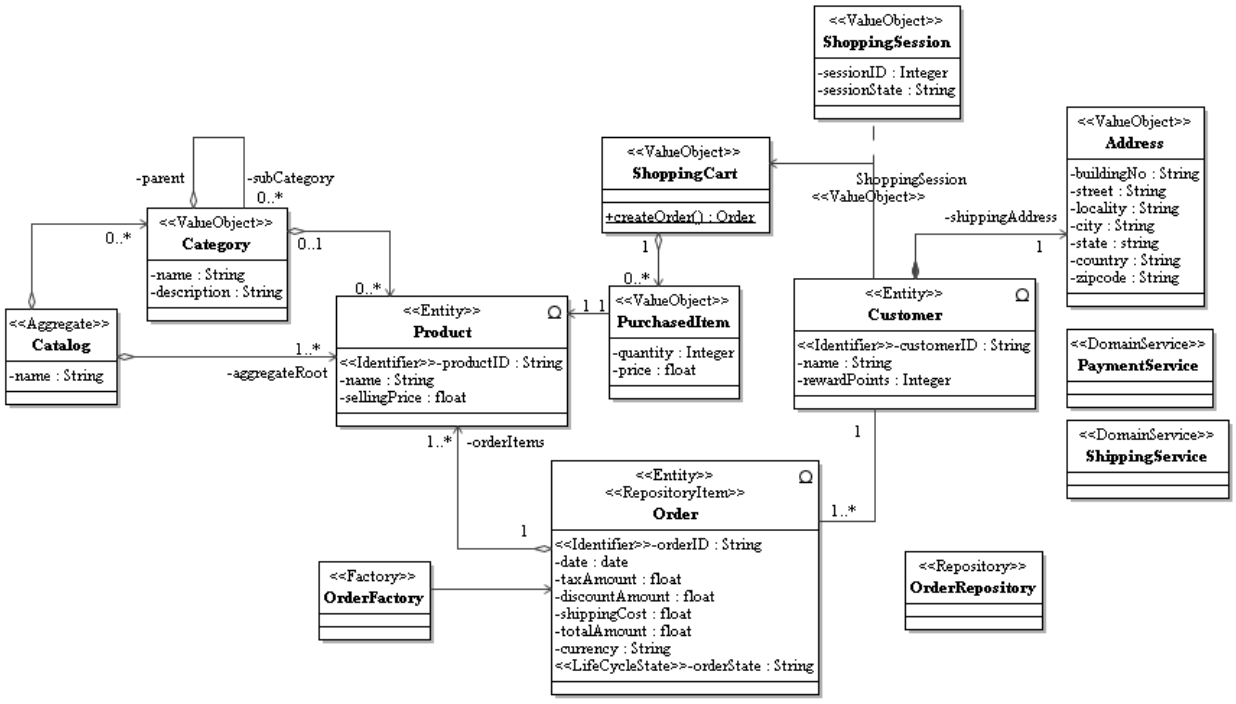


Fig 3.11: Online Shopping Domain Model (partial)

We use a fictitious online shopping scenario based on eBay ProStores®[4] to illustrate the use of our DDD Metamodel and the Resources Metamodel, to model web resources. The ProStores® platform supports a merchant to setup a web store to sell products on the internet. By using an iterative domain driven design approach, we arrive at the online shopping domain model (fig 3.11) modeled using our DDD Metamodel. We only present the partial domain model for brevity. Using the domain model, it is possible to generate the skeleton provisioning code in a language of choice. For our example, we use Java and

[4] The ProStores® scenario we describe here is fictitious, however it closely relates to the real world online selling scenario supported by ProStores®.

hence using standard JET (Java Emitter Template) transformations, we create partial provisioning code (fig 3.12) below:

```
package com.fictitious.prostores.ordermanagement;

public class Product implements RepositoryItem{
    private boolean isAggregateMember = false;
    private String UUID;

    private String productId;
    private String name;
    private float sellingPrice;

public boolean checkConsistency(){
    boolean consistencyCheckFailed = false;
    /*
     * Check for Consistency
     */
    return consistencyCheckFailed;
    }
}
```

```
package com.fictitious.prostores.ordermanagement;

import java.io.Serializable;

public interface RepositoryItem extends Serializable {

}
```

```
package com.fictitious.prostores.ordermanagement;

import java.util.*;

public class Order implements RepositoryItem{
    private boolean isAggregateMember = false;
    private String UUID;

    private String orderId;
    private Date date;
    private float taxAmount;
    private float discountAmount;
    private float shippingCost;
    private float totalAmount;
    private String currency;
    private String orderState;

    private List<Product> orderItems;
    private Customer customer;

    public boolean checkConsistency(){
        boolean consistencyCheckFailed = false;
        /*
         * Check for Consistency
         */
        return consistencyCheckFailed;
        }
}
package com.fictitious.prostores.ordermanagement;

public class Customer implements RepositoryItem{

    private boolean isAggregateMember = false;
    private String UUID;

    private String customerId;
    private String name;
    private Integer rewardPoints;

    public boolean checkConsistency(){
        boolean consistencyCheckFailed = false;
        /*
         * Check for Consistency
         */
        return consistencyCheckFailed;
        }
}
```

```
package com.fictitious.prostores.ordermanagement;

import java.util.*;

public class OrderRepository {

    public Order findByIdentifier(String UUID, String OrderID){
        Order foundOrder = new Order();
        /* Code here */
        return foundOrder;
    }
    public List<Order> findByCriteria(Order criteria){
        List<Order> foundOrders = new ArrayList<Order>();
        /* Code here */
        return foundOrders;
    }
    public boolean add(List<Order> items){
        /* Code here */
        return true;
    }
    public boolean remove(List<Order> items){
        /* Code here */
        return true;
    }
    public boolean update(List<Order> items){
        /* Code here */
        return true;
    }
}
```

Fig 3.12: Java Code Listing - Provisioning

Now consider that the platform owner has a new requirement to support. The scenario could be that, a merchant wants to support his customers to place an order directly in the ProStores® platform. The customers would want to place orders from his home-grown procurement application instead of the web shop shopping cart. To support this scenario, the platform owner could open up the 'Order' (domain) entity as a web resource which could be manipulated by a CRUD pattern by an external consumer. By doing so, an 'Order' can be created directly in the ProStores® platform. Fig 3.13 shows the resource model derived from the domain model using the model-to-model transformation. We have a XML representation of the resources in our example.

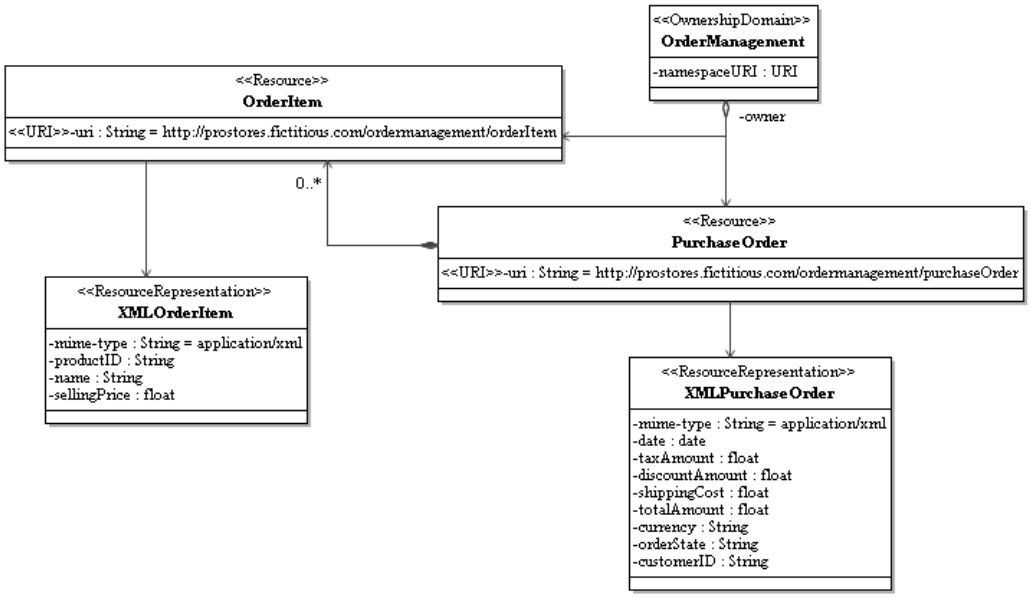


Fig 3.13: Online Shopping Resource Model

The resource could either be exposed through the SOAP-style or REST-style services. Fig 3.14 shows an abstract MANAGEPURCHASEORDERSERVICE WSDL 2.0 description. The WSDL service description is obtained from the resource model using a model-to-text transformation. If the platform owner chooses to expose the resource based on the REST-style, we provide a WADL (Web Application Description Language) description (fig 3.15) of the service using a resource model to WADL model-to-text transformation. WADL provides a machine process-able description of REST-based services.

```
<?xml version="1.0" encoding="utf-8" ?>
- <description xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://prostores.fictitious.com/services/ordermanagement"
    xmlns:tns="http://prostores.fictitious.com/services/ordermanagement"
    xmlns:order="http://prostores.fictitious.com/schemas/ordermanagement"
    xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions">
    <documentation>Manage Order Service</documentation>
  - <types>
    - <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://prostores.fictitious.com/schemas/ordermanagement"
        xmlns="http://prostores.fictitious.com/schemas/ordermanagement">
      + <xs:complexType name="OrderItemType">
      + <xs:complexType name="PurchaseOrderType">
        <xs:element name="PurchaseOrder" type="PurchaseOrderType" />
        <xs:element name="PurchaseOrders" type="PurchaseOrderType" minOccurs="0"
          maxOccurs="unbounded" />
      + <xs:element name="StandardException">
      </xs:schema>
    </types>
  - <interface name="ManagePurchaseOrderServiceInterface">
      <fault name="StandardFault" element="order:StandardException" />
    + <operation name="CreatePurchaseOrder"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri">
    + <operation name="ChangePurchaseOrder"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri">
    + <operation name="DeletePurchaseOrder"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri">
    + <operation name="QueryPurchaseOrderById"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri">
    + <operation name="QueryPurchaseOrderByCriteria"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri">
    </interface>
</description>
```

Fig 3.14: MANAGEPURCHASEORDERSERVICE – *SOAP-style Interface*

```
<?xml version="1.0" ?>
- <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://research.sun.com/wadl/2006/10/wadl.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:shopping="http://prostores.fictitious.com/schemas"
    xmlns="http://research.sun.com/wadl/2006/10">
  - <grammars>
    - <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://prostores.fictitious.com/ordermanagement/schemas"
        xmlns="http://prostores.fictitious.com/schemas/ordermanagement">
      + <xs:complexType name="OrderItemType">
      + <xs:complexType name="PurchaseOrderType">
      + <xs:element name="StandardException">
        <xs:element name="PurchaseOrder" type="PurchaseOrderType" />
        <xs:element name="PurchaseOrders" type="PurchaseOrderType" minOccurs="0"
          maxOccurs="unbounded" />
      </xs:schema>
    </grammars>
  - <resources base="http://prostores.fictitious.com/orgermanagement/">
    - <resource path="PurchaseOrder">
      + <method name="PUT" id="CreatePurchaseOrder">
      + <method name="POST" id="ChangePurchaseOrder">
      + <method name="POST" id="DeletePurchaseOrder">
      + <method name="GET" id="QueryPurchaseOrderById">
      - <method name="GET" id="QueryPurchaseOrderByCriteria">
          <request />
          <response />
        </method>
      </resource>
    </resources>
  </application>
```

Fig 3.15: WADL description – Purchase Order

3.5 Summary

In this chapter we presented our approach to model resources as well as fine-grained services to manipulate these resources. In our approach, any resource could be uniformly manipulated through CRUD-interfaces. Our modeling approach is based on domain-driven design with the focus on domain models. From the domain models, we derive the resources model depicting web resources. The transformation from a domain entity to a resource is based on business requirements, as illustrated in the online shopping scenario. The resource model is in the domain-layer in our simplified 4-layered SOA architecture, while the fine-grained services to manipulate resources are in the service-layer. The fine-grained access to web resources could either be through SOAP-style interface or through a REST interface. We support this through transformation of the model to executable

specifications using model-to-text transformations. The choice of style is left to the web platform owner.

In chapter 4 we address modeling of coarse-grained services in the service-layer. These services are modeled from a stake-holder perspective. Some of these services could be directly provisioned using the *Domain Services* we described in the DDD metamodel.

Chapter 4

Modeling Coarse-Grained Services

Model Views and a Services Metamodel to model different facets of Coarse-Grained Services

In this chapter, we address coarse-grained services. Modeling these services involves capturing various service requirements and solutions identified during early-stage services development using high-level conceptual models. Earlier, we argued that in order to improve the longevity of the solution and to rigorously represent all facets of services, it is important to capture the solution space in a platform and technology independent way using high-level models. These models must be rich enough to capture associated services metadata irrespective of whether the current standards support them. These models must support business users to visually model services. In this chapter, we identify different perspectives of services modeling and present six model views to support different facets of services. We also define our MOF2-based Services Metamodel to support modeling these different perspectives. Our Services Metamodel is a layer 2 (M2) model and extends the *UML Infrastructure Library::Core* package (herein known as Core) (fig 4.1).

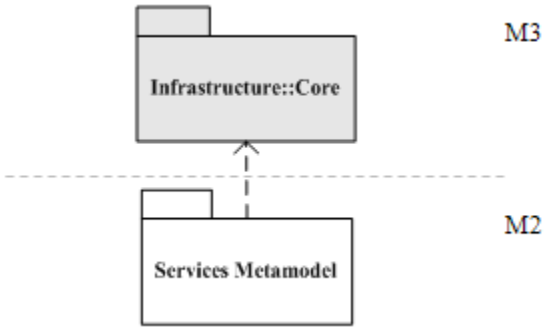


Fig 4.1: MOF2-based Services Metamodel

4.1 Services Metamodel – High-Level Requirements

Our Services Metamodel addresses the early-stage modeling of web-based electronic services from the perspective of web business platforms. A Services Metamodel must not only enable capturing of different perspectives of services, but also support maximum expressiveness with a small set of modeling elements (see criteria #3). Some of the criteria we mentioned in section 1.3 are addressed by the high-level requirements for the Services Metamodel we present below:

R1: The metamodel shall enable capture of the high-level description of the service (addresses criteria #1 – high-level conceptual service representation).

R2: The metamodel shall enable capture of the different roles and perspectives of the actors associated with a service (addresses criteria #4)

R3: The metamodel shall enable capture of realization (or provisioning) of services.

R4: The metamodel shall enable capture of the operational details of a service in use.

R5: The metamodel must be conformant to the Oasis SOA Reference Model (SOA-RM)

Some of our high-level requirements also correspond to a subset of mandatory requirements in the OMG's RFP (Request for Proposal) – the UML Profile and Metamodel for Services (herein referred to as RFP-UPMS) [96]. Though our intention is not to address all the requirements in RFP-UPMS, we just provide a correlation. The high-level description of a service (R1) includes ownership information, service capabilities and the roles involved in exercising these capabilities. Service interfaces and their operations, along with their pre-conditions and post-conditions, must also be a part of the service description. These correspond to RFP-UPMS mandatory requirement Service Description (requirement 6.5.7). The metamodel must support different roles of actors associated with the service (R2). Roles may include those of providers, consumers, aggregators and mediators. These requirements correspond to the RFP-UPMS mandatory requirements Service Provider and Service Consumer (requirements 6.5.12, 6.5.13). The metamodel must support realization mechanisms of services (R3). The realization mechanisms could include implementation by service providers or through aggregation of already existing

services by an aggregator. These correspond to the RFP-UPMS mandatory requirements Realization, Composition, and Extension (6.5.14, 6.5.15, 6.5.17). The metamodel must support provisioning of services (R4) over many channels, deployment and invocation mechanisms for the service. These correspond to the RFP-UPMS mandatory requirement Invocation (6.5.9). In addition, our metamodel also meets the requirements on UML Compatibility (6.5.2) and Platform Independence (6.5.3).

4.2 Services Model Views

A model view is a representation of one aspect or perspective of a system [97]. By looking at a system through different viewpoints, we would be able to deal with different aspects of the system better. In order to arrive at the right set of model views for modeling services for web business platforms, we look at a practical example. Our example is based on a real life scenario – eBay® Auctions[5] [98]. eBay® is a web-based business platform which allows auctioning of a variety of items based on certain rules and policies in an online marketplace. Sellers can auction items by choosing a minimum bid amount and duration. Bidders bid for the item and the bidder with the highest bid at bid closing wins. The winning bidder pays the seller and the seller ships the item to the buyer. Both the buyers and sellers rate each other and the rating determines their credibility in the marketplace. eBay® supports business services such as auctioning, bidding and rating but collaborates with business partners for payment (Paypal®) and shipping services (FedEx®). There could as well be other partners in the services marketplace providing these services.

eBay® opened-up its ecommerce web business platform for customers, business partners and affiliates by exposing their business functions as webservices-based web APIs. Opening up of the platform would enable a manufacturing company to auction its excess inventory through eBay® auctions directly from its Supplier Relationship Management (SRM) software like SAP® SRM. To expose a business function as a consumable service, a business expert must be able to specify the broad definition of this service in a technology-agnostic fashion. Consider the AUCTIONITEM service which allows sellers to list an item in eBay® auctions. A business expert from eBay® needs a model view for

[5] The services and the scenario described here are completely fictitious. However the scenario is representational of other web business platform scenarios.

defining this service, its broad purpose and the associated ownership domain. *An ownership domain represents a logical partitioning of the services for administrative or deployment purposes.* We need a ***Service Definition View*** to support the business expert in defining a service. The service definition view must support classification of the services as atomic, composite or abstract. Atomic services (e.g. AUCTIONITEM) represent atomic business functions such as auctioning an item. Composite services aggregate other services (constituent services) and through this packaging improve value proposition to consumers. Consider the BUYITNOW service from eBay® which lets a seller directly sell the item at a fixed price instead of auctioning it. The BUYITNOW service in turn uses the PROCESSPAYMENT service from Paypal® and SHIPPINGSERVICE from FedEx®. The BUYITNOW service which aggregates the PROCESSPAYMENT and the SHIPPINGSERVICE services provided by business partners is a composite service. Lastly, a business expert must be able to define a service that represents a business need – a gap in the value-chain – yet to be provided by any service provider. The reasons for defining an abstract service are the following:

- An ownership domain would like its business partners to provide it with this service based on some terms and conditions (expression of intent to outsource the service)

- The ownership domain would want to defer the realization of the service. The service could be defined for advertisement purposes but need not necessarily be realized immediately.

For example, assume that eBay® wants to introduce a new VALIDATEAUCTIONITEM service which would validate certain items being auctioned (e.g. art pieces). eBay® would want to outsource the realization of this service (to say artnet®) because they might not have the expertise to do this in-house. The intention to outsource this service could be expressed by defining an abstract service.

The next step after service definition for the AUCTIONITEM service would be to define the capabilities provided by this service more concretely. How does the interface for auctioning an item look like? What are the different service properties (e.g. cost of access, availability etc.)? We need a ***Service Capability View*** which would define service properties and capabilities. The view must describe service interfaces, their service operations and the syntax associated with invoking these operations along with the schema of the messages and the message exchange pattern for interacting with the service. The service capability view essentially defines the *capability on-offer* of a service.

Once the basic capabilities and properties are defined, it must be possible to specify policies (such as security policy, auction policy or service disruption policy). The security policy could state that only registered and authorized users must be allowed to access the AUCTIONITEM service. The auction policy could state that perishable items could not be auctioned. We need a ***Service Policy View*** to define policies on services. Exact mechanisms to realize these policies are specified by the IT team during realization by enhancing the policy definitions. For example, the IT experts could decide that the authorization (security policy) should happen through a signed certificate (e.g. X.509 digital certificate [99]). The service policy view defines the *terms of offer* of a service.

The AUCTIONITEM service once defined and capabilities expressed must be realized (or provisioned). Realizing a service could either be through an existing or new implementation – in case of atomic services (by service providers) or through service composition – in case of composite services (by service aggregators). The IT team could use underlying IT assets from operational systems (packaged applications, custom home-grown systems or mainframes) to implement a service. We need a ***Service Realization View*** to capture provisioning of services.

Every service consumer has a goal which a service offering would satisfy. The relationship between consumer goals and service offerings is an *n x m* relationship. Sometimes there may not be a direct 'fit' between the goals and services due to inherent heterogeneities, resulting in the need for mediation. For example, the BIDFORITEM service which is used to bid for an auctioned item could be re-purposed to support a proxy-bidding scenario. In a proxy-bidding scenario, the system alters the bid for an item on behalf of the bidder based on user-specified rules. A service mediator could support this proxy-bidding scenario. We need a ***Service Mediation View*** to support specification of mediation.

After the abstract definition for a service is specified followed by the service realization, it must be possible to define external interaction points through which a service consumer could access the service. It must be possible to define various ways of binding to the service through the use of different transport protocols. It must also be possible to define service invocation properties. We need a ***Service Deployment View*** to describe the service interaction points and service invocation mechanisms.

From the eBay® Auctions scenario, we have identified the six model views – service definition view, service capability view, service policy view, service realization view, service mediation view and the service deployment view. These six views represent different perspectives of services modeling in the context of web business platforms. Table 4.1 below provides a summary of our six services model views.

Model View	Viewpoint addresses
Service Description View	Description and classification of Services based on ownership domain
Service Capability View	Description of Service, Service Properties, Interfaces, Operations, Messages and message-exchange pattern. In essence, defines the capability on-offer.
Service Policy View	Definition of Service Policies. In essence, defining the term of offer of a service
Service Realization Service	Defining the service provisioning approach, either service implementation from underlying IT assets or through composition of constituent services
Service Mediation View	Defining how existing services could be re-purposed to address different consumer goals using process or data mediation.
Service Deployment View	Describes service interaction points and service invocation mechanisms

Table 4.1: Summary of Services Model Views

4.3 Formal Semantics of the Services Metamodel

In this section, we provide the formal semantics of the Services Metamodel which supports modeling in different views. In addition to the MOF2-based metamodel for each model view, we also present UML2 Profile defined in a stereotyped package <<*Modeling View*>> for each of these model views which would help leveraging existing UML tools. In this section, we focus on five model views and we deal with the service policy view separately in the next chapter.

4.3.1 Service Definition View

The service definition view (fig 4.2) defines a service, its purpose along with the ownership domain which owns the service. The ownership domain provides a logical partitioning of services in terms of physical or administrative boundaries. The business entity which owns the service could be the top-level ownership domain. Enterprise-wide service portfolio could be organized under hierarchies of ownership domains. Ideally, the business expert uses the service definition view as a starting point to define the ownership domain and the services they own.

Key Classes and Associations

Service: A service represents a capability of a provider which meets goals of consumers. It is a first-class modeling entity in our Services Metamodel. Service extends the *Core: NamedElement* from UML Infrastructure Library[6]. A service could be atomic, composite (isComposite = true) or an abstract service (isAbstract = true).

Ownership Domain: An ownership domain represents partitioning of services based on physical deployment or administrative domains [32]. Ownership domain has owned services associated with it. An ownership domain can in turn belong to other ownership domain thereby creating a hierarchy of ownerships. The ownership domain extends the *Core: Namespace*. The ownership domain also has a namespace URI (uniform resource identifier).

[6] *Core* represents the UML Infrastructure Library – the *kernel*. In our diagrams, we use the stereotype <<*metaclass*>> to represent the Core.

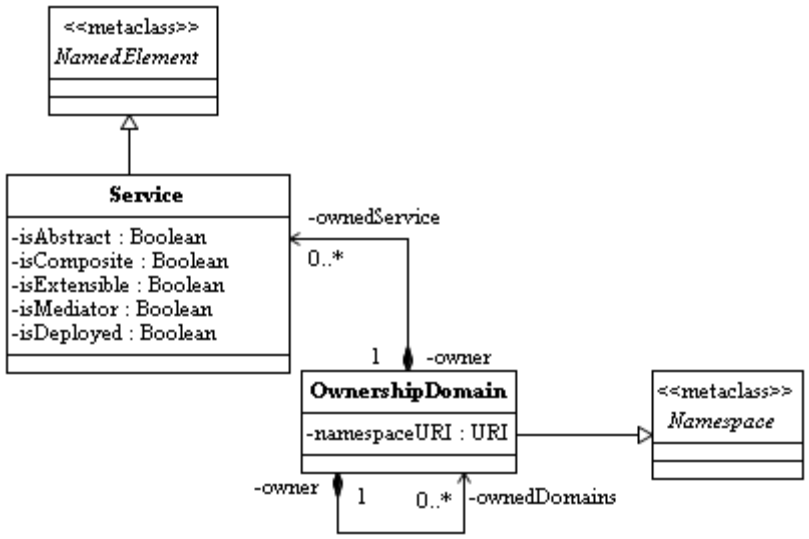


Fig 4.2: Service Definition View

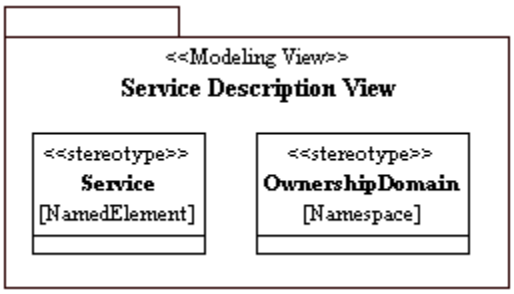


Fig 4.3: Service Definition Modeling View

4.3.2 Service Capability View

The service capability view (fig 4.4) helps in defining the capabilities and properties of a service which is defined using the service definition view. Using this view it is possible to define the service description, service properties, the service interface and the various service operations along with their pre- and post-conditions.

Key Classes and Associations

Service: Service has a property *isExtensible* which determines if the service could be extended or enhanced. Extension of a service could either be functional enhancements (extending its capabilities) or property enhancements (enhancing service properties or policies associated with a service). Every service can have one or more service descriptions.

Service Description: A service description has a semantic description of the service and could have many classifications. Service Description is associated with a Service Property and a Service Interface. A service could have multiple service descriptions; also a service description could exist without any service realizing it.

Service Property: A service property represents properties of service such as cost of access, availability and service rating. The property could be quantitative (describing a measure) or qualitative. One or more service properties are associated with every service description. Service properties are also used in identifying appropriate services during service discovery.

Service Classification: A service could be classified based on different taxonomies. The classification system could be based on an internal taxonomy or be based on an existing system such as the North American Industry Classification System (NAICS) [100]. It extends the *Core: Namespace*.

Service Interface: A service interface represents the underlying capabilities brought to bear by a service. A service interface could be extended to support specialization of a service. For a service to be extended, the *isExtensible* property must be set *true*. Every service interface has a set of supported operations and an exception associated with it. The Service Interface extends the *Core: NamedElement*.

Service Constraint: A service constraint represents pre-conditions or post-conditions on service operations. A constraint could be a hard constraint (mandatory constraint) or just a preference (best-effort constraint). Service Constraint extends the *Core: Constraint*. A service constraint is owned by an OwnershipDomain.

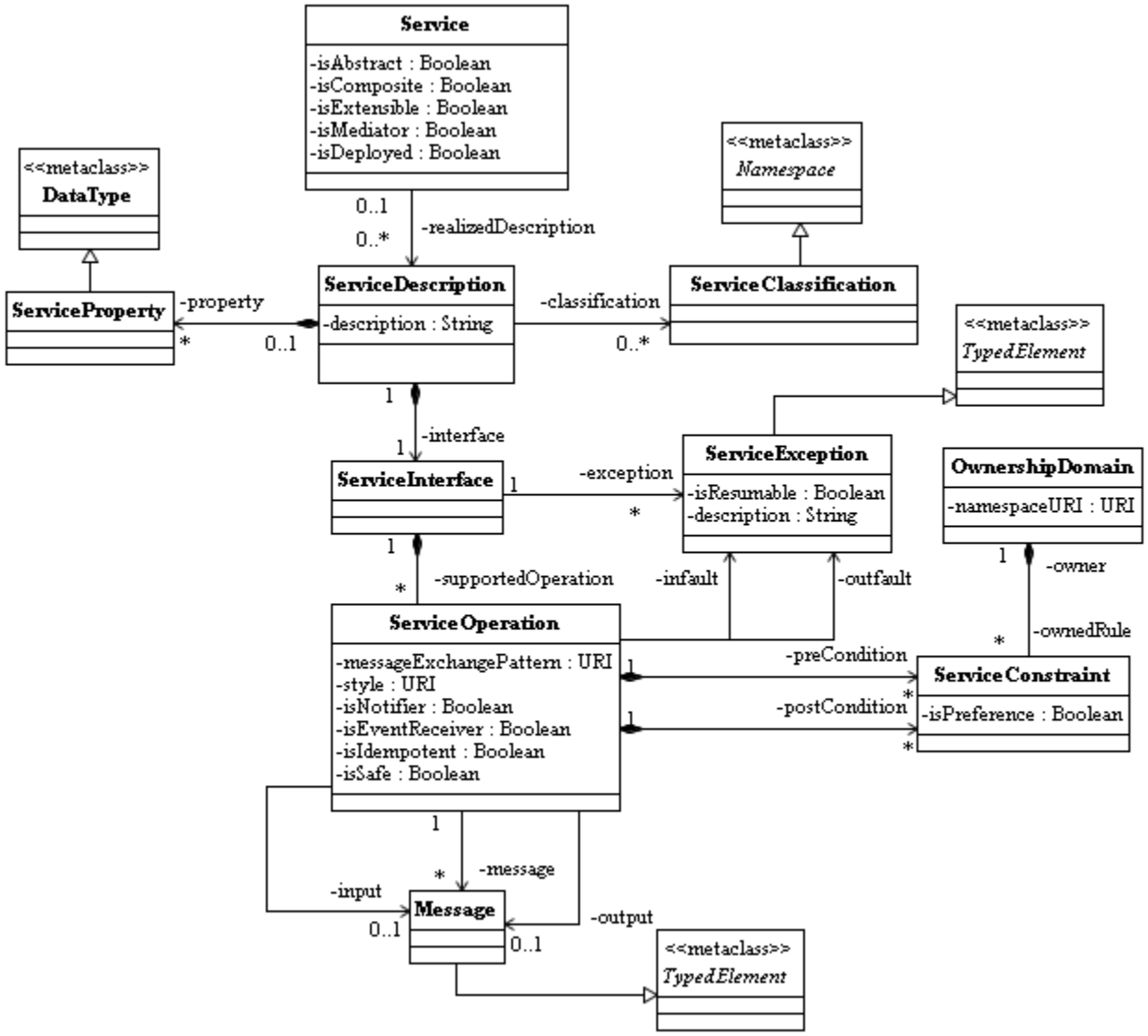


Fig 4.4: Service Capability View

Service Operation: A service operation represents an underlying capability. Event-driven scenarios could also be modeled using a notification or an event receiver operation. A notification operation (*isNotifier = true*) sends out messages that represent a notification, whereas an event receiver operation (*isEventReceiver = true*) receives messages representing an event. Every operation has input and output messages and the

sequencing of these messages get determined by the message exchange patterns. Marking an operation as *isNotifier* or *isReceiver* could determine the message exchange pattern. It extends *Core: NamedElement*.

Service Exception: A service exception represents an exceptional condition in a service operation execution. Every service operation would have an *infault* or an *outfault* message based on the message exchange pattern. A message exchange pattern defines the order of the messages between the provider and the consumer. An exception could also be defined at the level of a service interface. A service exception could be a *resumable* exception – an exception does not halt the further execution of an operation after being handled properly. It extends the *Core: TypedElement*.

Message: A message encapsulates input and output data for a service operation. We use the terminology *message* as it is indicative of a loosely-coupled communication between service providers and consumers. The messages that are exchanged must be strictly typed and hence Message extends *Core: TypedElement*. The message label identifies whether a message is an input message or an output message.

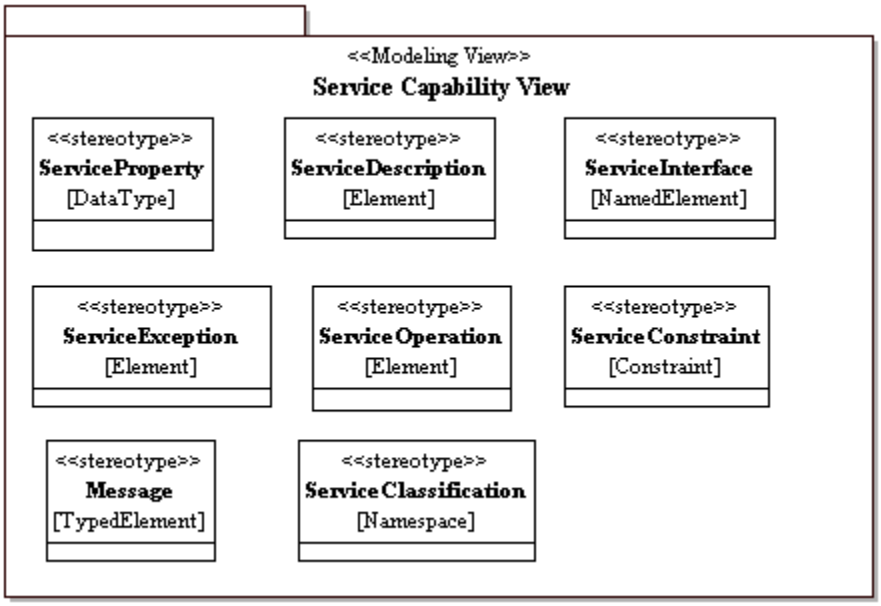


Fig 4.5: Service Capability Modeling View

4.3.3 Service Realization View

The service realization view (fig 4.6) helps to describe how services specified using the service definition, capability and policy views are realized. Service realization is done by IT experts. Realization of a service could be either through implementation or through composition. Atomic services are realized through service provider implementations whereas composite services are realized through composition of existing services. These service providers could be existing IT assets in the operational systems or new implementations. Composition is achieved by service aggregators using known composition patterns and composition directives. Design-time composition directives enable dynamic composition decisions at execution-time.

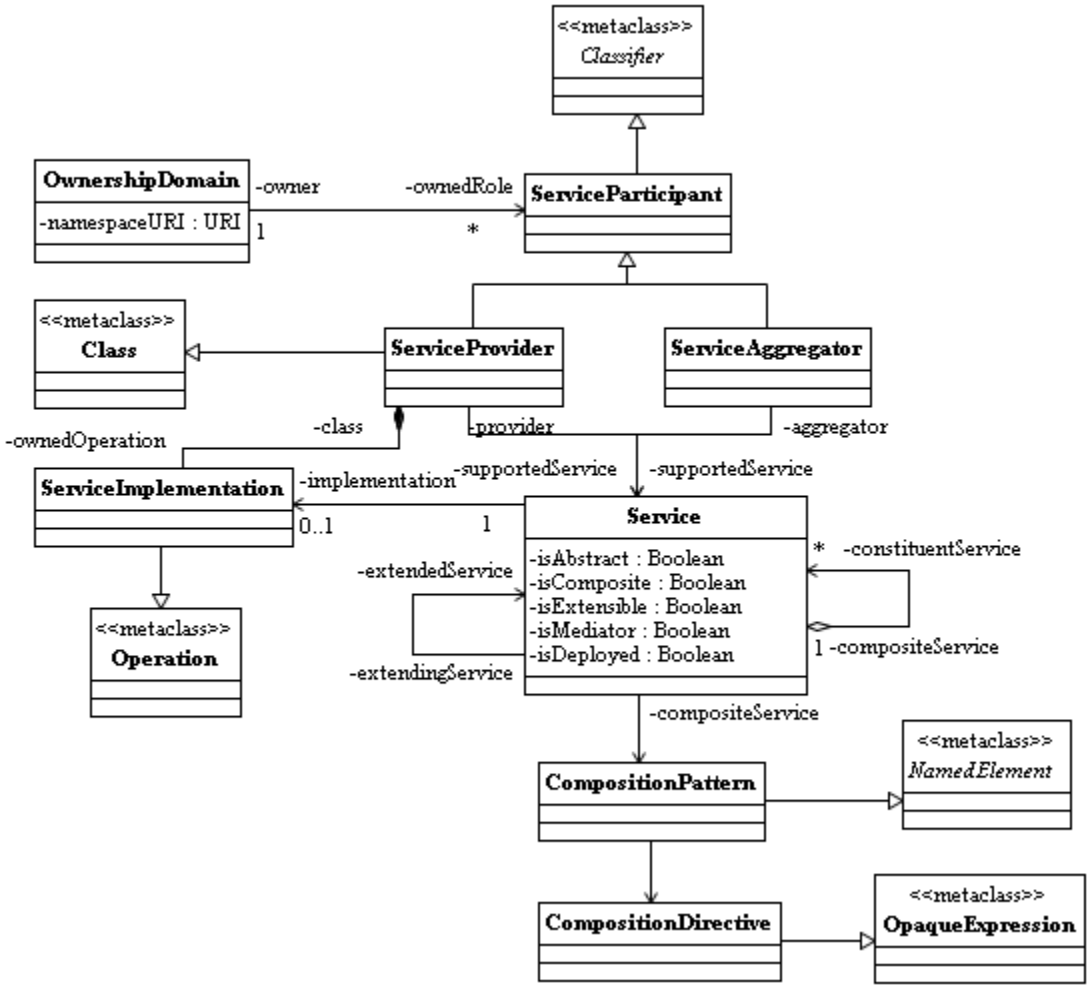


Fig 4.6: Service Realization View

Key Classes and Associations

Service: Service has an attribute *isComposable* which determines if the service is composable. Also if a service is a composite service, then it is composed of many constituent services. Each composite service has an associated composition pattern.

Service Participant: A service participant represents a role played by a stakeholder in a services marketplace. Service Provider, Service Aggregator, Service Consumer and Service Mediator are different roles representing service participants. It extends the *Core: Classifier*.

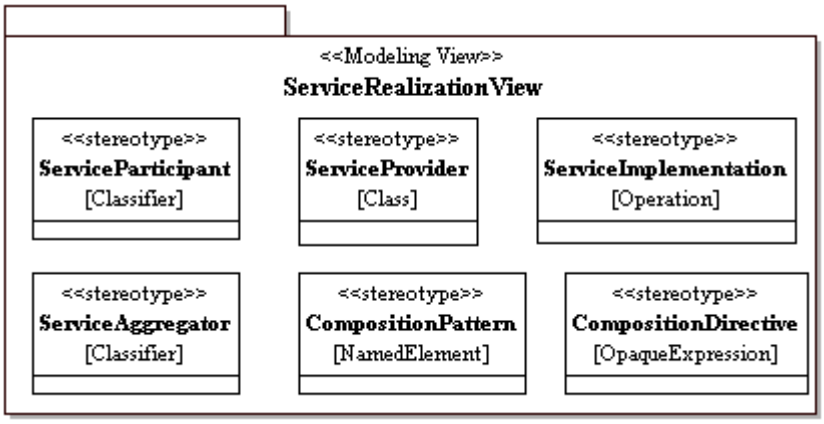


Fig 4.7: Service Realization Modeling View

Service Provider: A service provider supports realization of an atomic service through its service implementations. A service could be realized with an existing off-the-shelf component, a function module in a packaged application, a stored procedure in a database or through an entirely new implementation. A service provider is a service participant and it also extends the *Core: Class*.

Service Implementation: A service implementation extends the *Core: Operation*. It supports the actual implementation of an atomic service.

Service Aggregator: A service aggregator aggregates different (constituent) services to provide a value-added composite service through service composition. A service aggregator is a service participant.

Composition Pattern: A composition pattern is a pattern which describes a structured assembly of constituent services to create a composite service. There are many patterns available in the literature. The composition pattern extends the *Core: NamedElement*.

Composition Directive: The composition directive represents a directive used for composing constituent services to create a composite service using the composition pattern. A composition directive is associated with a composition pattern. It extends the *Core: OpaqueExpression*.

Constraints

1. Only Composite services have a composition pattern associated with them.

2. Only Composite services have an aggregator associated with it.

3. The supported service of a service provider is always an atomic service

4. The supported service of a service aggregator is always a composite service

5. If one or more constituent services of a composite service are abstract, then the constituent service is also abstract.

4.3.4 Service Mediation View

In a loosely-coupled environment, mediators are needed to cope with inherent heterogeneities. Service Mediators are used to re-purpose services to cater to a wider variety of user goals. Such mediation is called process mediation. Service Mediation is also needed during service composition to support differences in service data and message schemas. Such mediation is called data mediation. The Service Mediation View (fig 4.8) helps in defining data or process mediation scenarios.

Key Classes and Associations

Service Mediator: A service mediator facilitates mediation for a service. Mediation could either be data or process mediation. The type of mediator is specified by MediatorType (data or process). The mediatorType attribute denotes the actual mediation. The service mediation provides a mediation service which supports the actual mediation.

Service: A service has an attribute *isMediator* which signifies whether a service is a mediator or not. The mediator service can either mediate between a consumer and a service or between a service and another service. It also has another attribute *isRealized* which determines whether the service has a realization.

Goal: The goal represents the goal (or need) of a service consumer. Each Service Consumer has associated goals (consumer goals). Each of these goals is satisfied by one or more services (satisfying services) and each service supports one or more goals. A goal has both pre- and post-conditions which have to be met if the goal has to be satisfied. Goal extends *Core: NamedElement*.

Service Mediation: Service Mediation represents the mediation between either two services (in a composition scenario) or between a service and an external service consumer. A service mediator is associated with a mediator service which does the actual mediation. It extends the *Core: DirectedRelationship*.

Constraints

Abstract services are not realized (*isRealized* = false).

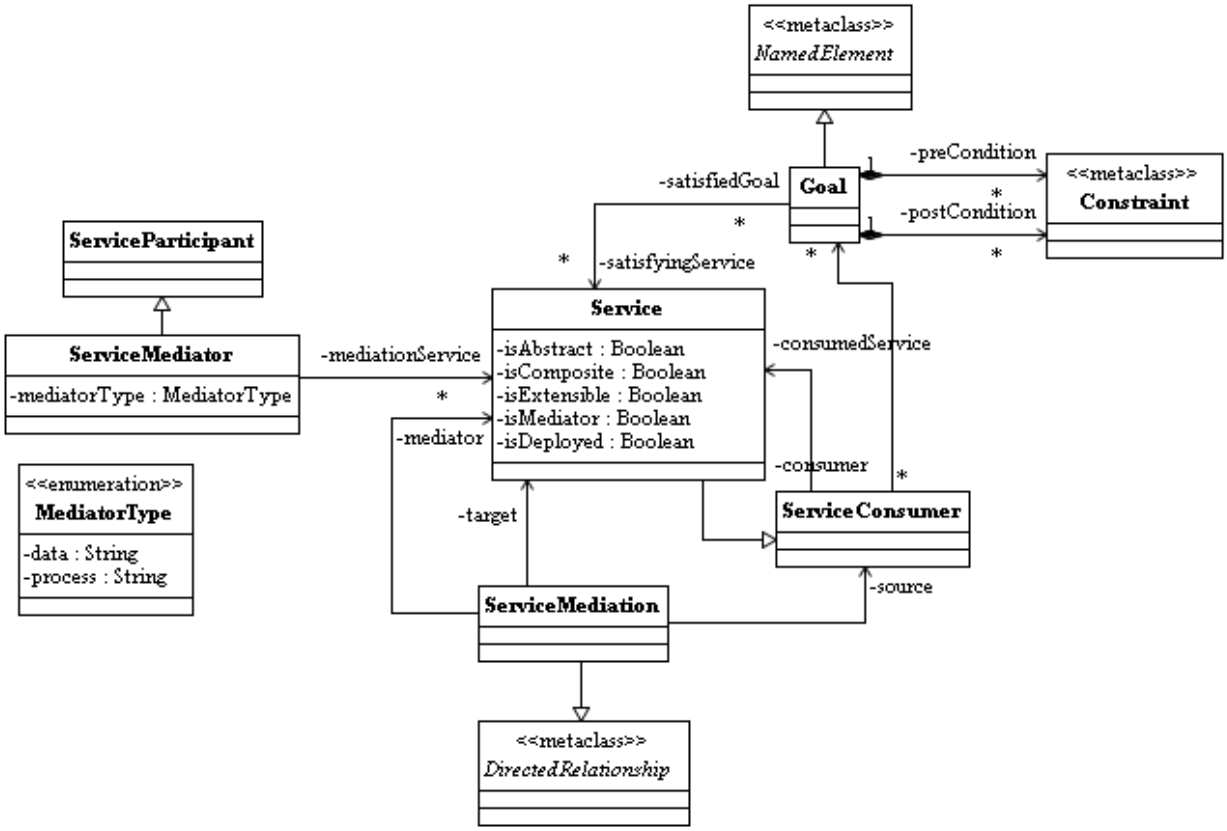


Fig 4.8: Service Mediation View

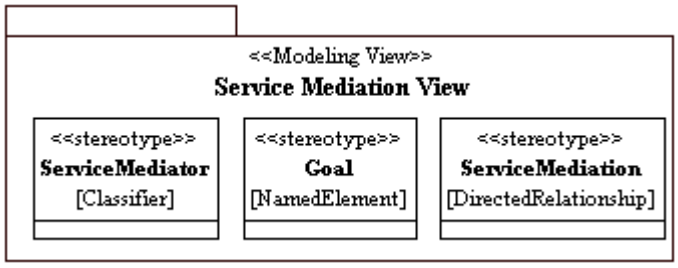


Fig 4.9: Service Mediation Modeling View

4.3.5 Service Deployment View

The service deployment view (fig 4.10) helps to describe how realized and concrete services are deployed and how they could be invoked by external stakeholders.

Key Classes and Associations

Interaction Point: The interaction point defines an endpoint at which a service could be accessed by service consumers. It is uniquely determined by a location URI. Interaction point encapsulates the semantics of a channel through which a service is exposed (exposed service). The choice of a channel is represented by a BindingType i.e. logical channel type such as SOAP, HTTP etc. An ownership domain may have one or more interaction points. Also a service could be exposed through different interaction points (end-points). It extends *Core: NamedElement*.

Service Invocation: Service invocation defines an invocation of a service through the interaction point by an external service consumer or another service (in a service composition scenario). It also defines the mode of interaction i.e. Invocation Mode – whether the service invocation is synchronous or asynchronous. A service invocation could be either stateful (*isStateful* = *true*) or stateless.

Service: Service has an attribute *isDeployed* which signifies if a service is deployed and has at least one interaction point.

Constraints

Abstract services will not have interaction points since they cannot be deployed (*isDeployed* = false).

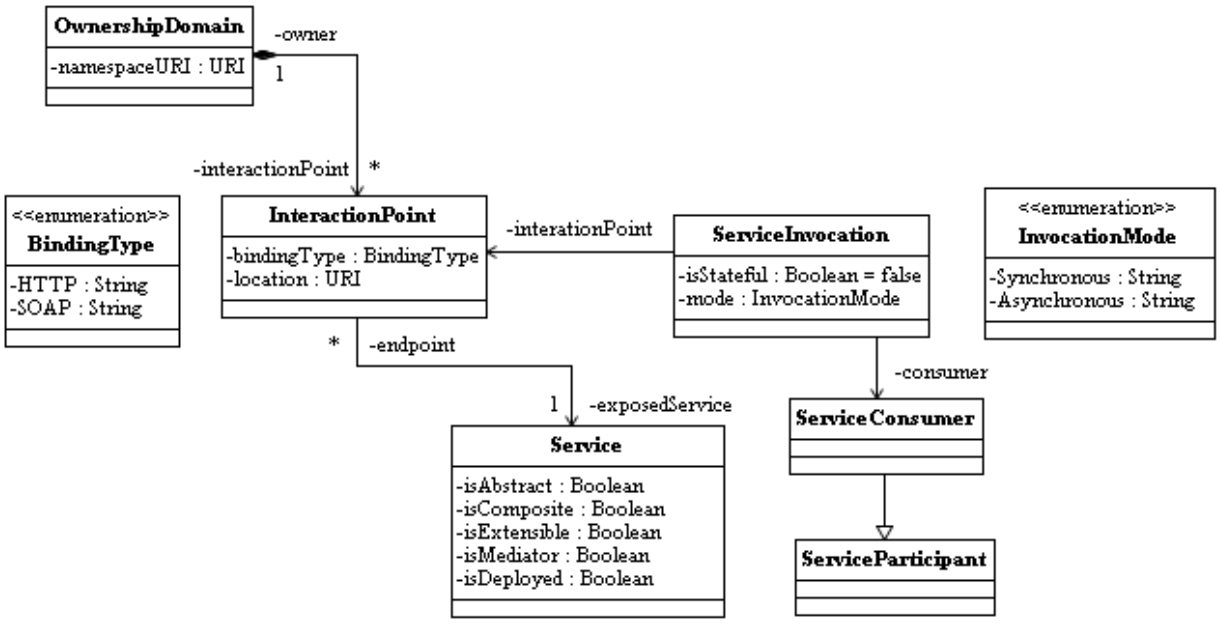


Fig 4.10: Service Deployment View

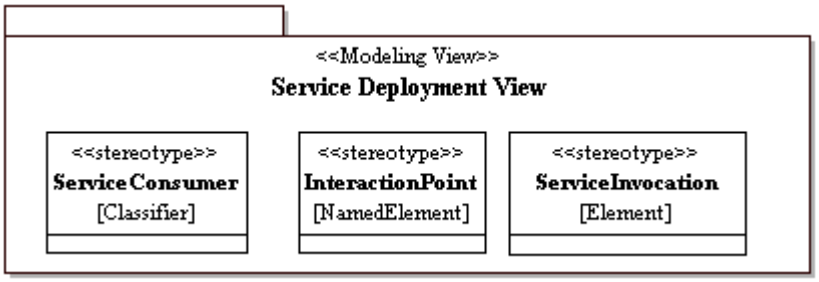


Fig 4.11: Service Deployment Modeling View

4.4 Modeling the Internet Auctions Scenario

In this section, we use our Services Metamodel to model the eBay® internet auctions scenario, particularly we would keep the focus on simple modeling of the AUCTIONITEM service in four steps. We use this modeling exercise to demonstrate the usefulness of our metamodel.

Step I: Defining the AUCTIONITEM Service

We start with the service definition view to define all the services and the ownership domains in the internet auctions scenario (see fig 4.12). The eBay ownership domain owns the BUYITNOW composite service (*isComposite = true*). It also owns other ownership domains – Feedback and Auction ownership domains, which in turn own atomic services such as RATINGSERVICE, AUCTIONITEM and BIDFORITEM. The Paypal ownership domain owns the PROCESSPAYMENT service and the FedEx ownership domain owns the SHIPPINGSERVICE.

Step II: Modeling the AUCTIONITEM Service Capability

We take the AUCTIONITEM service and model its *capability on-offer* (see fig 4.13) using the service capability view. The service description has a single classification based on the NACIS. The AUCTIONITEM service is classified as a business-to-consumer (B2C) service. The service description has an AuctionServiceInterface $^{(SI)}$ which has an AuctionNotPermitted $^{(EX)}$ service exception. The exception denotes a business contract violation of trying to auction a prohibited item (could be based on the auction policy). The service interface supports a service operation AuctionSingleItem $^{(OP)}$ which follows the request-response message exchange pattern. Since the pattern supports an *in* as well as an *out* message, we have defined both the messages. The AuctionRequestMessage$^{(M)}$, the *in* message encapsulates item name, item description, minimum bid and auction closing date and AuctionResponseMessage$^{(M)}$, the *out* response message returns an auction identifier as a reference.

Step III: Realizing the AUCTIONITEM Service

The AUCTIONITEM service is an atomic service that is realized (or provisioned) by a service provider implementation (see fig 4.14). The auction manager is the service provider which provides a service implementation *listAuctionItem ()* which implements the service.

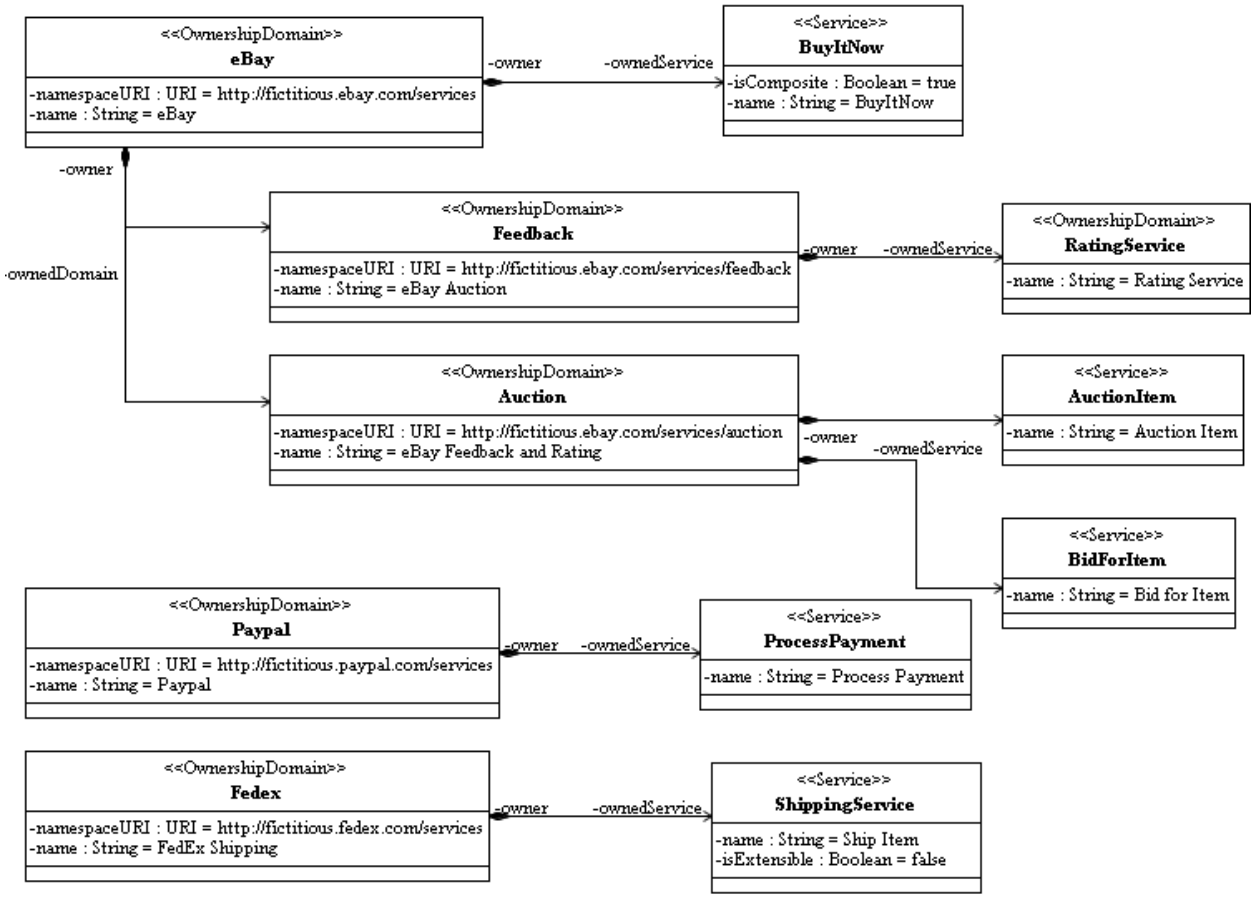


Fig 4.12: eBay Internet Auctions Scenario - Service Definition

Step IV: Deploying the AUCTIONITEM Service

Once the AUCTIONITEM is defined, capability modeled and realized it has to be exposed (or deployed) for consumption (see fig 4.15). We expose the AUCTIONITEM service through an interaction point AuctionSOAPEndpoint $^{(IP)}$. The transport protocol for this endpoint is SOAP/HTTP. The location URL is specified through which this service could be consumed synchronously by a seller from the public domain.

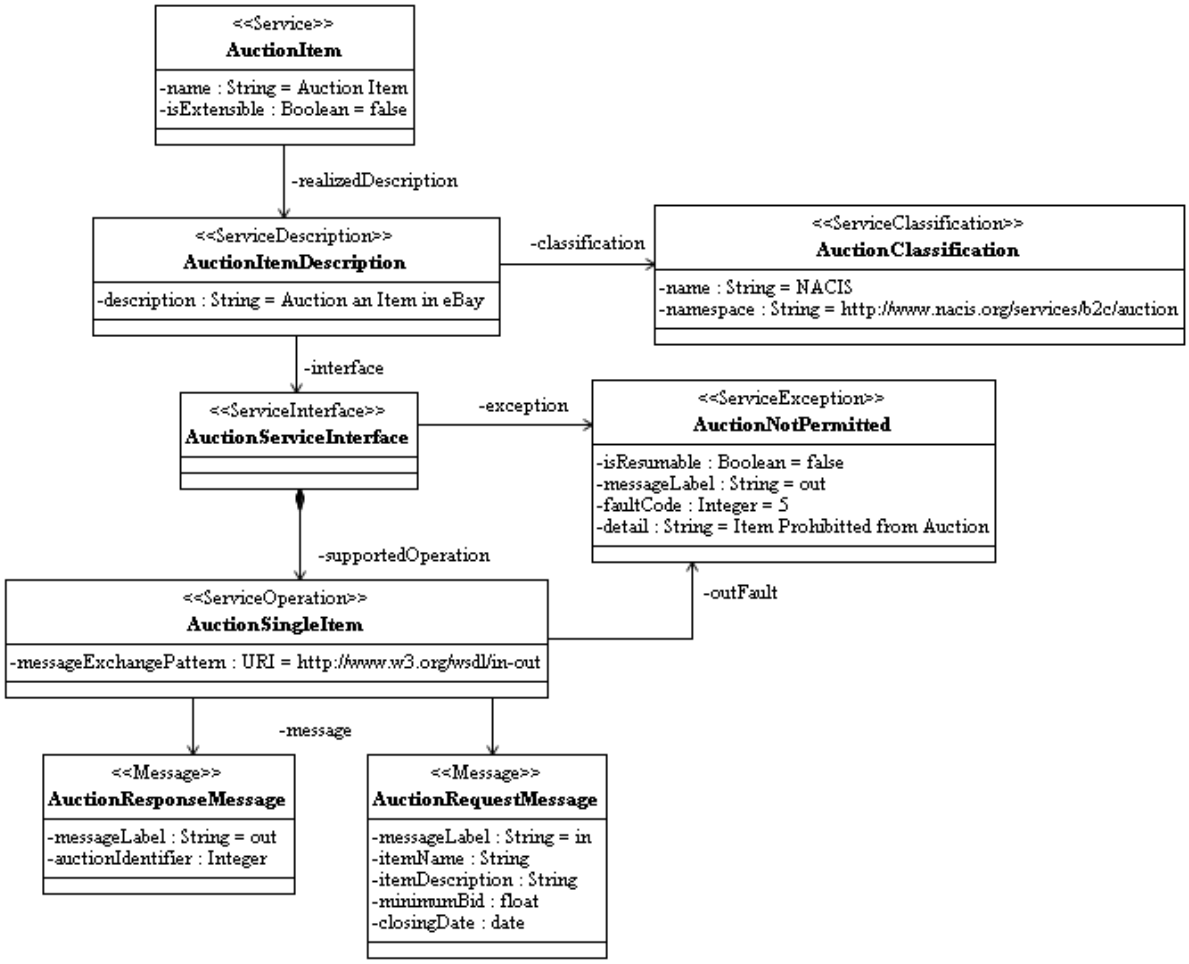


Fig 4.13: AUCTIONITEM *service - Service Capability*

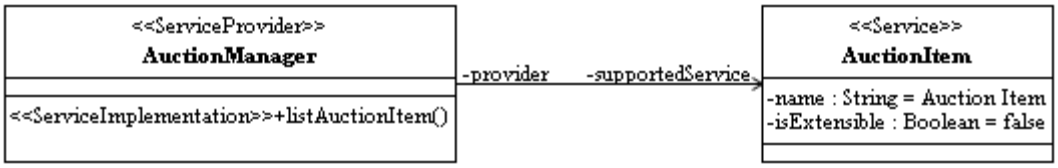


Fig 4.14: AUCTIONITEM *service - Service Realization*

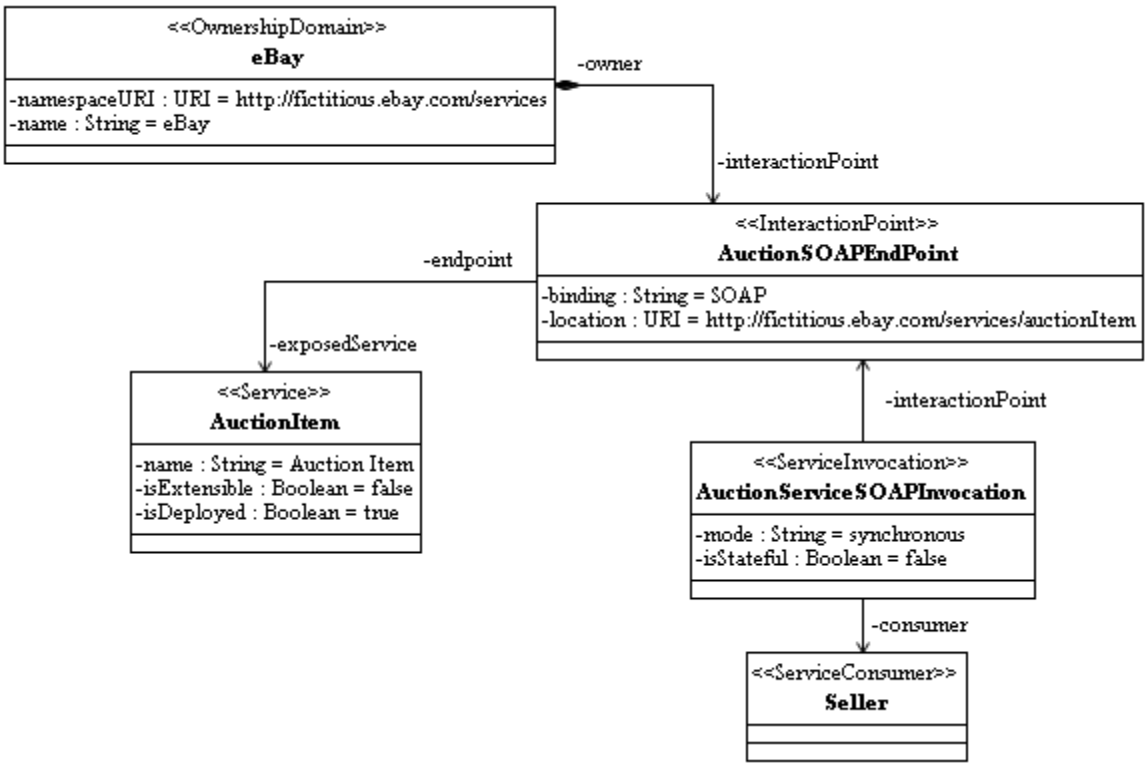


Fig 4.15: AUCTIONITEM *service - Service Deployment*

4.5 Models to Executable Specifications

Once the service is modeled using our model views and our Services Metamodel, we would have to transform these models to usable artifacts. The artifacts which get generated from our services models are listed below (table 4.2).

Input Models	Specifications
Service Description Model / Service Capability Model	- Abstract Service Description (e.g. WSDL) – service description without concrete protocol bindings and endpoints - Service registry listings (e.g. UDDI [31]) for aiding service discovery for consumers

Service Realization Model	Service provisioning code templates based on particular provisioning environments (java, .NET etc.)
Service Policy Model	Policy descriptions (e.g. WS-Policy [101]) and policy attachment to services (e.g. WS-PolicyAttachment [102]) (more in chapter 5)
Service Deployment Model	- Concrete service descriptions (WSDL with protocol bindings) - Deployment descriptors

Table 4.2: List of Artifacts Generated from Models

Generally, these artifacts could be executable specifications, deployment descriptors as well as code templates. In model-driven development, the transformations from model to text (artifacts) happen through model to text transformation languages. Model to text transformation is evolving and there are multiple transformation languages such as MOFScript, OMG's Model to Text Transformation Language (MTL), JET (Java Emitter Templates) and OpenArchitectureWare's Xpand [103]. The choice of the languages could be based on MDA tools, personal preferences and suitability (for example, JET is used to create transformations between model to Java code). The support for model transformation is one of the core features which support longevity and usefulness of a MDA solution.

For each of these artifacts generated, there could be one or more input models. Using the service description and the capability models, we could generate abstract service descriptions (e.g. WSDL without bindings). We could also create service registry entries to aid consumers in discovering the service using standards such as UDDI. Using the service realization model, we could create code templates to support service provisioning.

For example, we could create a service provider class to support the provisioning of the AUCTIONITEM service. The class template could be created for different technical platforms such as java or .NET. The service policy model could be transformed to run-time policy specifications (more in chapter 5) and the deployment model could be used to create concrete service specifications with protocol bindings and endpoint. It could also be used to create application server deployment descriptors. Technically, we could have transformations from our Services Metamodel to any executable service description

language. But our choice of executable specification for our default transformation is driven by industry adoption. Based on industry adoption, Web Services Description Language (version 1.1 or 2.0) is the widely adopted service description standard. We have developed a MTL template (see Appendix I) to transform our services models to WSDL 2.0 service description (fig 4.16). For provisioning the AUCTIONITEM service, the service realization model is converted to a skeleton java class (fig 4.17). It is also possible to create the provisioning class in any technology platform by altering the transformation.

```
<?xml version="1.0" encoding="utf-8" ?>
- <description xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://fictitious.ebay.com/services/auction"
    xmlns:tns="http://fictitious.ebay.com/services/auction"
    xmlns:aus="http://fictitious.ebay.com/schemas/auction"
    xmlns:wsoap="http://www.w3.org/ns/wsdl/soap" xmlns:soap="http://www.w3.org/2003/05/soap-
    envelope" xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions">
    <documentation>Auction an Item in eBay</documentation>
  - <types>
    - <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://fictitious.ebay.com/schemas/auction"
        xmlns="http://fictitious.ebay.com/schemas/auction">
      + <xs:element name="AuctionRequestMessage">
      + <xs:element name="AuctionResponseMessage">
      + <xs:element name="AuctionNotPermitted">
      </xs:schema>
    </types>
  - <interface name="AuctionServiceInterface">
      <fault name="AuctionNotPermitted" element="aus:AuctionNotePermitted" />
    - <operation name="AuctionSingleItem" pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri">
        <input messageLabel="In" element="aus:AuctionRequestMessage" />
        <output messageLabel="Out" element="aus:AuctionResponseMessage" />
        <outfault ref="tns:AuctionNotPermitted" messageLabel="Out" />
      </operation>
    </interface>
  - <binding name="AuctionSOAPEndpoint" interface="tns:AuctionServiceInteface"
      type="http://www.w3.org/ns/wsdl/soap"
      wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
      <fault ref="tns:AuctionNotPermitted" wsoap:code="soap:Sender" />
      <operation ref="tns:AuctionSingleItem"
        wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response" />
    </binding>
  - <service name="AuctionItem" interface="tns:AuctionServiceInterface">
      <endpoint name="AuctionItemEndpoint" binding="tns:AuctionSOAPBinding"
        address="http://fictitious.ebay.com/services/auctionItem" />
    </service>
  </description>
```

Fig 4.16: AuctionService WSDL 2.0 Description

```
package com.fictitious.ebay.auctions;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

 /* Generated Service Provider Class
  * Implemented Service: AuctionItem
  */
public class AuctionManager {

    private class AuctionItemResponse{
        private Integer auctionIdentifier;
    }
    private class AuctionItemRequest{
        private String itemName;
        private String itemDescription;
        private float minimumBid;
        private Date closingDate;
    }


/*
 * <<ServiceImplementation>> listAuctionItem
 * Description: Supports the provisioning of atomic service AuctionItem
 */
public List<AuctionItemResponse>
       listAuctionItem(List<AuctionItemRequest> request){
    List<AuctionItemResponse>
            response  = new ArrayList<AuctionItemResponse> ();
    //insert implementation here
    return response;
    }
}
```

Fig 4.17: Auction Manager Java Implementation – Service Provisioning

4.6 Related Work

Model-driven development of services is still in the nascent stage. Model-driven development of web-services is addressed in [104-107]. The RFP-UMPS is an effort by OMG to consolidate existing approaches into a consistent metamodel and UML2 profile for modeling services. There are existing UML-based approaches to modeling services. In [108], they use UML class diagrams to model services. In this approach 'Service' is not viewed as a first-class modeling entity. UML Collaboration diagrams have been used

extensively to model behavioral-aspects such as service collaboration and compositions [109,110].

There are also other efforts to provide support for services modeling through light-weight extensions to UML through Profiles [111-113]. All these efforts provide a direct mapping between WSDL 1.1 elements and their model elements. Also they are based on the UML1.x standards. UML-profiles for services and SOA are proposed by [114,115]. An UML 2.0 Profile for Software Services [116] is proposed by IBM. In this profile, a service is restrictively modeled as a Port of a UML Composite class. The service realization mechanisms are only through implementation by components. Composition as a realization mechanism is not supported. The profile does not support modeling of policies and mediation. Although Service is a first-class modeling entity, it is tightly associated with a Component. In contrast, in our services metamodel services are truly first-class modeling entities. Modeling of realization mechanisms such as implementation and composition are supported. Our service metamodel also supports modeling non-functional aspects of services through service properties and policies. We also provide support for deployment and mediation of services.

UML-profile for distributed object computing (EDOC) [117] facilitates modeling of enterprise systems but does not provide means to model services. The UN/CEFACT's Unified Modeling Methodology (UMM) [118] provides a standard way for business processes and information modeling for e-Commerce. An UML-profile for B2B e-commerce is presented in [119]. Our services metamodel could complement these approaches and act as the foundation for a model-based service repository.

Apart from UML-based modeling approaches, there are other approaches which aid modeling of services. [120] provides a formal-model of services with a theoretical foundation for specifying services and service composition. The Webservices Modeling Framework (WSMF) [121] defines conceptual entities for services modeling. Web-Service Modeling Ontology (WSMO) [122] has its foundations in WSMF but it defines a formal ontology to semantically describe webservices. The Webservices Modeling Language (WSML) [123] provides a formal syntax for WSMO based on different logical formalisms.

4.7 Summary

In this chapter, we presented our Services Metamodel with six model views to model different perspectives of services development. These model views have a formal foundation based on MOF2. They support different stakeholders such as business experts and the IT experts to model services during early-stage services design. The metamodel draws its foundations from technical specifications like WSDL 2.0, WS-Policy and WSMF since our focus is on web-based electronic services. We have used our services metamodel to model a fictitious eBay® auctions scenario. Through this modeling exercise, we have demonstrated how different facets of services such as service description, realization, mediation and deployment could be modeled using our services metamodel. The Services Metamodel addresses the high-level requirements we had mentioned. The service description and capability views address the high-level description of web-based services. Each of the views addresses different participants (roles) – service provider, aggregator, mediator and consumer, involved. The service realization view addresses the provisioning of defined services. The service deployment view addresses operational details of the service such as available interaction points, protocols and modes to access the service.

Chapter 5

Modeling Service Policies

Modeling the *Terms of offer* of a Service using Service Policies

A service representation describes two facets of a service – the service functionality (*capability on-offer*) and the terms at which the service is offered (*terms of offer*) [33]. The capability on-offer satisfies the goal of a service consumer under the constraints of the terms of offer. Essentially, the terms of offer describe the service-level agreement (SLA) between a service provider and a consumer. Service Policies are used to define the terms of offer of a service offering. In general, a service policy defines constraints or conditions of use of a service. Policies deal with different aspects such as security, pricing, quality of service etc. Consider the example of a SHIPPINGSERVICE from FedEx®, the capability on-offer is to ship packages from one place to another, the terms of offer could be the time-to-delivery and rates of shipping. In our research, we address two significant issues in the development of service policies.

- Firstly, current approaches to service policies focus primarily on technical or infrastructural aspects such as security, trust and reliable messaging. We take a broader view of service policy development. In our view, service policies would address three-levels of aspects – *service-level* (e.g. availability, pricing, promotions and quality of service), business or *domain-level* (e.g. compliance, industry regulations) and *technical-level* (e.g. security, trust). While technical policies are defined by IT experts, the service-level and domain policies would be defined by domain experts.

- Secondly, we address independent development of service policies. Traditionally, service descriptions have had a bias towards describing service functionality as opposed to non-functional terms of offer (e.g. WSDL for web service description). Lately, there have been efforts to address description of non-functional terms of offer in service descriptions (Features & Properties in WSDL 2.0 and the WS-Policy framework [124]). However, service development approaches still consider service

policies in the confined context of the underlying service functionality which they constraint. Instead, service policies could be developed independently by domain experts and could later be applied on a chosen set of services in the services portfolio through well-defined quantification and fine-tuning. For example, the security expert could define encryption and authentication policies independently and later apply it to selected services in the portfolio.

5.1 Generic Policy Framework

The most-important aspect of our service policy development approach is our service policy view supported by the service policy metamodel – a part of our Services Metamodel. In order to arrive at our policy metamodel for the service policy view, it is important to understand the generic policy model – an abstract model for service policies. The generic policy model consists of four functional layers to describe service policies [125] (fig 5.1).

- **Vocabulary Specification Layer:** Deals with specification of the vocabulary associated with various policy domains representing independent aspects. These aspects could be technical, service-level or domain-level aspects. Vocabulary consists of vocabulary items and their applicable values which would then be used in service policies. It also involves specifying the semantics and syntax associated with the vocabulary items. Constraints are always specified on these vocabulary items in the constraint specification layer.

- **Constraint Specification Layer:** Deals with specification of policy constraints, which would ideally be constraints on the agreeable values of vocabulary items. Constrained vocabulary items are assertions which are the building blocks of a policy.

- **Policy Specification Layer:** Deals with specification of acceptable combinations of the constrained vocabulary items. Each combination of constrained vocabulary items represents a policy alternative.

- **Bindings Specification Layer:** Deals with specification of application of the service policies on various policy subjects. Policy subjects could be services,

ownership domains as well as individual operations. Binding layer supports the quantification and fine-tuning of policies for different policy subjects.

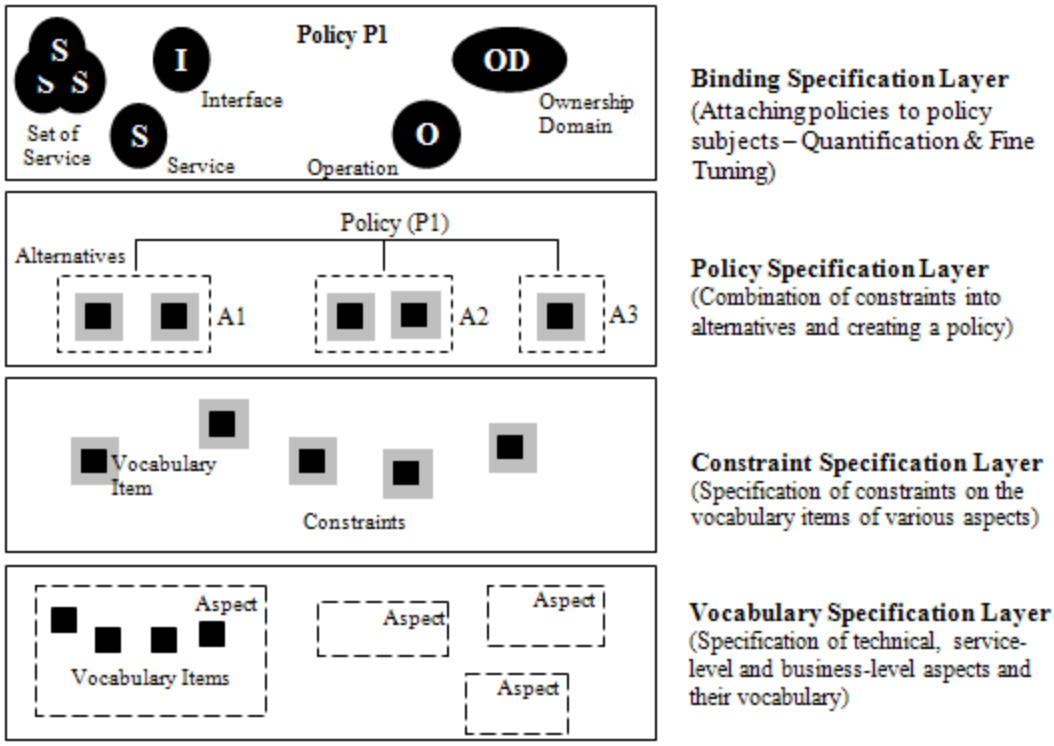


Fig 5.1: The Generic Policy Model

This generic policy model is largely representative of several policy specifications which are based on propositional logic with the assertions representing an indivisible unit and their combinations through conjunction or disjunction representing a policy.

5.2 Service Policy View

The service policy view (fig 5.2) is used to define policies which could later be applied on selected service artifacts. It is supported by the policy metamodel, part of the Services Metamodel. The service policy <<*modeling view*>> is presented in fig 5.3.

Service Policy: A service policy defines a set of enforceable constraints which would be applied on a policy subject [32]. It presents these enforceable constraints as a set of alternatives. A service policy reflects the point of view of a service participant. Service Policy extends the *Core: NamedElement*. A service policy is owned by an OwnershipDomain.

Policy Subject: A policy subject represents an entity on which a policy is applied [101]. A policy subject extends the *Core: Element*. The policy subjects could be Ownership Domain, Service, Service Interface, Service Operation, Message and Interaction Point (end point). If a set of policies are applicable on a single policy subject, at run-time these are reconciled and represented as an *effective policy*.

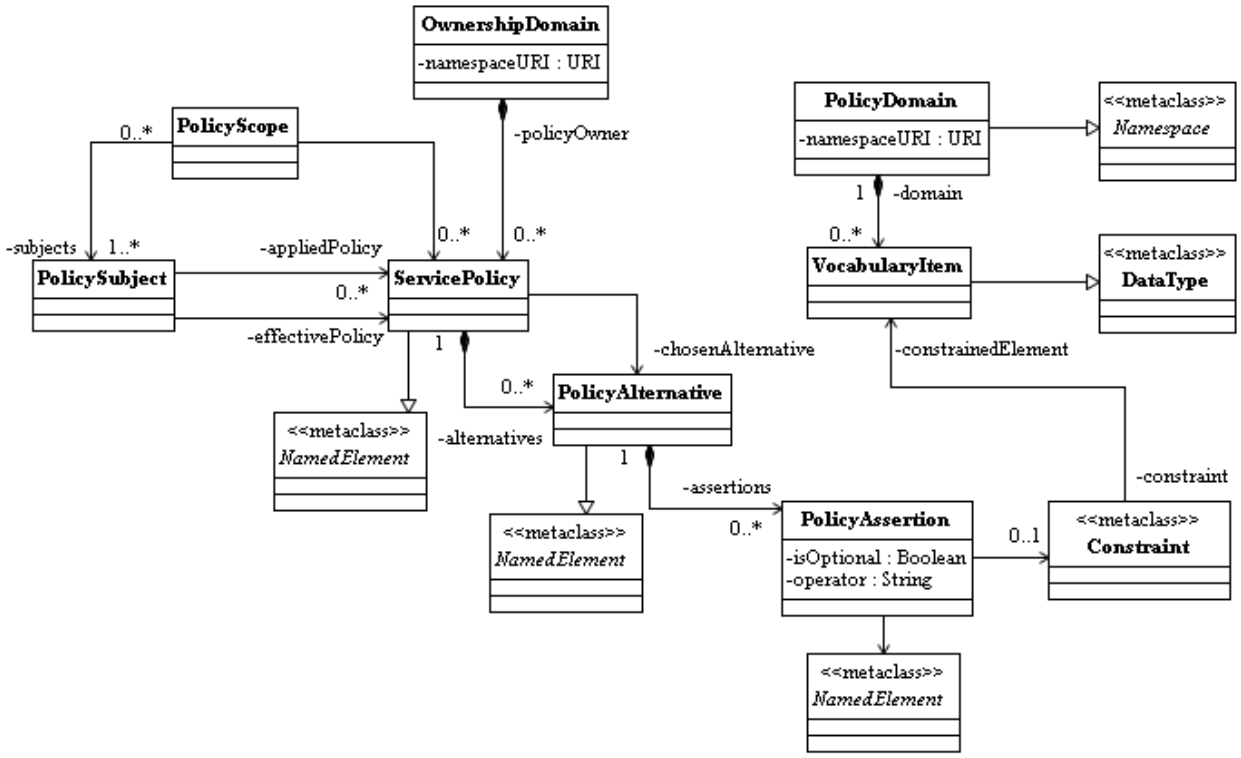


Fig 5.2: Service Policy View – The Policy Metamodel

Policy Scope: A policy scope represents a set of policy subjects on which a policy could be applied [101]. It is a mechanism to group related policy subjects together in order to apply the same policy on them. More than one policy could also be applied on the policy scope. The policy scope supports quantification of service policy by domain experts.

Policy Alternative: Each policy has a set of policy alternatives out of which at least one has to be honored [101]. The policy alternative which is honored is called the *chosen alternative*. Every policy alternative would have more than one policy assertion.

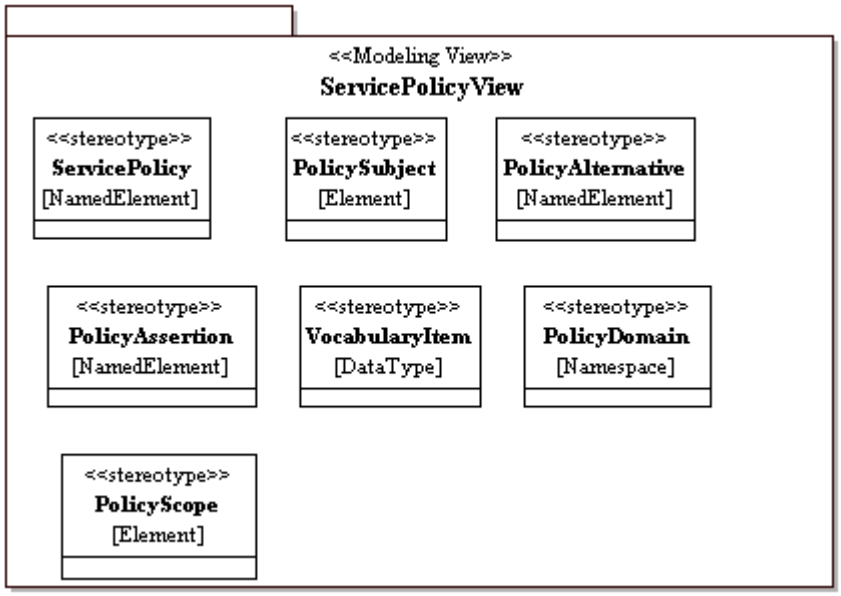


Fig 5.3: Service Policy Modeling View (UML Profile)

Policy Assertion: Every policy alternative would have one or more policy assertions [101]. A policy assertion is related to a constraint that is applied on a vocabulary item (constrained element) of a particular domain. The policy assertion specifies the allowable range, range of values, or set of values for a vocabulary item. It has an operator associated with it – the operator is a predicate operator used to describe constraints. The policy assertion could be optional in nature and could represent a preference of the service participant. It extends the *Core: NamedElement*.

A policy assertion is an atomic unit of a service policy. It represents a constraint on a vocabulary item representing different technical, service-level or domain-level (business) aspects. A policy assertion could be represented as:

Policy Assertion = {VI, PO, AV, C}

Where, VI - vocabulary item representing a particular aspect
PO - predicate operator
AV - accepted value / values or range of values
C – category of the assertion (Mandatory / preference)

Consider examples of security assertion (use of Kerberos security token) and pricing assertion (cost of service-access) below:

Security Token Assertion = {'Security Token', 'Equals', 'Kerberos', 'Preference'}

Pricing Assertion = {'Cost of Access', 'Equals', '1 EUR', 'Mandatory'}

Policy Domain: A policy domain represents a grouping of assertions belonging to a particular aspect such as pricing, availability, security & trust etc. A policy domain is identified by a name and a namespace URI and it extends the *Core: Namespace*.

Vocabulary Item: A vocabulary item represents semantics associated with a particular aspect and belongs to a policy domain. Every vocabulary item has a set of applicable values. The vocabulary items for a particular domain (aspect) are defined by the domain expert. Vocabulary Item extends the *Core: DataType*.

5.3 Vocabulary Specification – Defining Policy Domains and their Vocabulary

Vocabulary specification involves identifying policy domains and describing their vocabulary. The policy domain vocabulary involves defining *Vocabulary Items* to describe the policy domain. The vocabulary items would have a type and a range of acceptable values. The policy assertions apply constraints on the vocabulary items by specifying agreeable values for the vocabulary items. Policy domains address aspects that represent

independent concerns such as security, pricing etc. These concerns are pre-dominantly crosscutting in nature as they apply to a set of services in the services portfolio and not just a single service. From aspect-oriented software development literature, we refer to these crosscutting concerns represented by the policy domains as aspects. We group these aspects as *technical*, *service-level* or *domain-level* (business) aspects. It is important to identify these aspects early in the life-cycle of service development and define their vocabulary in order to use them in service policies.

5.3.1 Policy Domain Aspect Catalog

Becasue the technical, service-level and domain-level aspects are reusable assets in services development, it is important to document and catalog these aspects. Notably, this catalog of aspects is extensible and could be extended to create additional aspects either by extending existing aspects or by adding new aspects. We have defined a standard schema (table 5.1) to document aspects.

Name of Concern	The name of the concern addressed by the aspect
Type of Aspect	Denotes the aspect type
Related Aspects	Denotes related aspects for this aspect
Context	Denotes the context for this aspect
Rationale & Discussion	Provides a brief description of the aspect and its application
Quantification	Denotes applicability of the aspect. It could be: 1. List of services in the services portfolio 2. Select services, interfaces, operations or interaction points (end-points) 3. Ownership Domains

Vocabulary	Vocabulary defines a set of vocabulary items and their applicable values	
Vocabulary Items	Type	Applicable Values
Domain terms to describe the aspect	Type of vocabulary item	Acceptable values for the vocabulary item

Table 5.1: Standard Schema for Documenting and Cataloging Aspects

We believe this would facilitate better communication among stakeholders during early-stage design and development activities. A formal definition of this schema is done using XML-Schema (aspect.xsd). A pictorial XML-Schema is presented in fig 5.4.

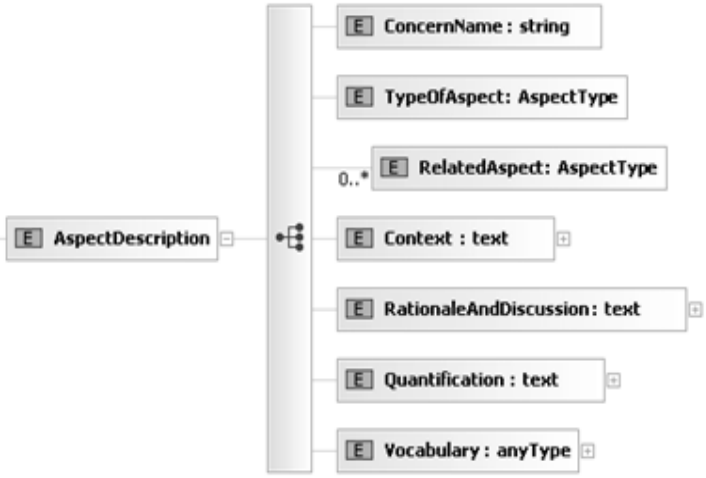


Fig 5.4: A XML-Schema (pictorial representation) for Documenting Aspects

In the remainder of this section, we present the top-level technical, service-level and domain-level aspects we have identified. An important point to note is that the vocabulary for these aspects would evolve and standardize over a period of time. Existing ontologies could also be used to standardize the vocabulary.

Technical Aspects

Technical aspects address infrastructural and messaging concerns such as security, trust and transactions. These aspects must be conveyed though service policies to enable secure trusted and reliable conversation between the service provider and the consumer. Fig 5.5 presents the top-level technical aspects we have identified – Security, Trust, Reliable Messaging and Transactions.

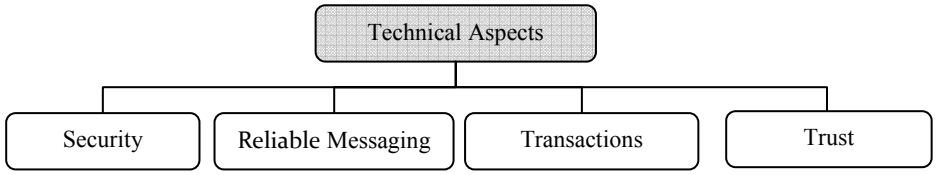


Fig 5.5: Technical Aspects

Security: Security deals with message level security between the service provider and consumer thereby guaranteeing a secure conversation. Security mainly involves end-to-end message integrity, message confidentiality and authentication. As an example, we use the catalog schema to document the security aspect (see Table 5.2).

Trust: Trust is closely related to security. In the context of a secure conversation, trust determines the reliability and integrity of the service consumer from the perspective of the provider or vice-versa. In order to prove integrity, the consumer requests a token from a trusted third-party (e.g. Kerberos token from a Kerberos Token Distribution Center) and sends this to the provider to establish it's identify.

Reliable Messaging: Reliable messaging deals with end-to-end reliable and guaranteed delivery of messages between a service provider and a consumer.

Transactions: Transactions addresses standard transaction mechanisms for short-duration ACID transactions as well as long-running business transactions.

Name of Concern	Secure Conversation	
Type of Aspect	Security	
Related Aspects	Trust	
Context	Security addresses secure conversation between the service provider and the consumer.	
Rationale & Discussion	Security addresses issues such as authentication, encryption and integrity of messages between the provider and the consumer.	
Quantification	Externally exposed services needing secure access	
Vocabulary		
Vocabulary Terms	Type	Applicable Values
Username	String	
PasswordType	String	Clear Text, Digest
PasswordValue	String	
IsBinarySecurityTokenRequired	Boolean	
BinaryEncodingType	String	Base64, Hex, UU
BinaryEncodingTokenType	String	Kerberos, X.509 (variants)
BinaryEncodingTokenValue	anyType	
isDigitalSignatureRequired	Boolean	
SignatureMethod	String	
HashMethod	String	SHA1, MD5
DigestValue	anyType	
IsEncryptionRequired	Boolean	

EncryptionMethod	String	DES, TripleDES, PGP

Table 5.2: Technical Aspect – Secure Conversation

Service-Level Aspects

Service-level aspects addresses service concerns such as quality of service, privacy of service consumers, pricing and availability. It also addresses how to promote the use of services in the services marketplace. Fig 5.6 describes the top-level service-level aspects we have identified.

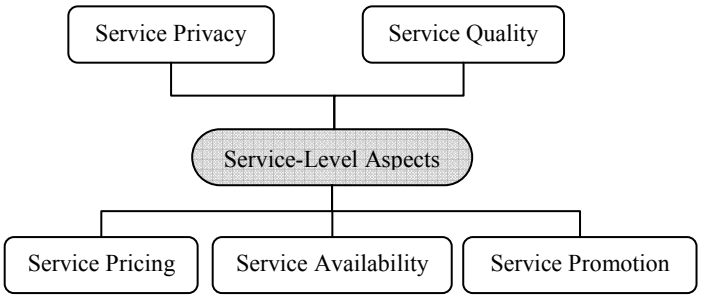


Fig 5.6: Service-Level Aspects

Service Pricing: Service pricing deals with the price at which a service is offered. It also deals with price types, payment modes and the charging styles for the use of a service. As an example, we use the catalog schema to document the pricing aspect (Table 5.3).

Service Availability: Service availability deals with spatial (location) and temporal availability concerns of a service. It determines the time of the day and the duration for which the service is available. It also determines the geographical reach (countries, regions, cities and states) of the service.

Name of Concern	Pricing of a Service Offering
Type of Aspect	Service Pricing
Related Aspects	Service Promotion
Context	A Service offering from a provider could have an associated cost.
Rationale & Discussion	Pricing deals with associating a cost of access with a service. It involves payment and settlement. Payment is a concern during access of a paid service. Payment for a service is determined by cost of service access, the charging style and the payment modes.
Quantification	Payment is a concern across a set of paid services.

Vocabulary		
Vocabulary Terms	Type	Applicable Values
Pricing Period	Validity	
Applicable Location	Location	
Pricing Mechanism	String	Absolute, Proportional, Dynamic
Price Amount	Amount	
Price Type	String	Regular, Tax, Shipping, Commission, Octroi
Credit Period	Duration	
Payment Mode	String	Cheque, Cash, Credit Card, Bank Transfer
Charging Style	String	Pay-per-use, Rental, Subscription

Table 5.3: Service-Level Aspect – Service Pricing

Service Promotion: Service promotion deals with promoting service consumption by customers and market segments by providing them with discounts and rewards.

Service Privacy: Deals with protecting consumer information and ensuring confidentiality of the data exchanged between the service consumer and the provider. It also determines whether the consumer information would be shared with business partners in case of composite service offerings.

Service Quality: Deals with guaranteeing consumers acceptable and agreed upon quality of service such as service availability, response time, performance and reliability.

Domain-Level Aspects

Domain-level aspects address business-level concerns such as compliance to legislative as well as industry regulations, adherence to business rules and following industry conventions (fig 5.7).

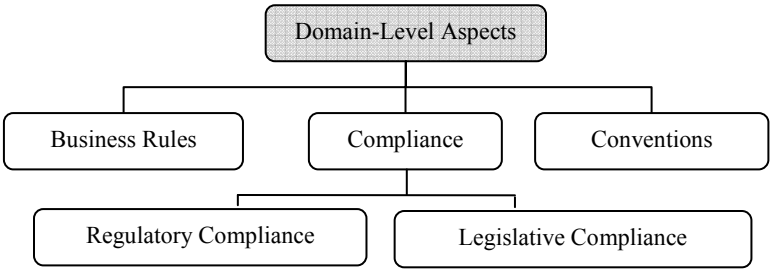


Fig 5.7: Domain-Level Aspects

Business Rules: Business rules define constraints on the operations, or operational procedure of a business which influences the behavior of the business. Business rules could pertain to business calculations, business policies or restrictions.

Compliance: Compliance addresses issues related to adhering to legislation (rule of the land) or with regulations set by industry regulatory authorities.

Conventions: Conventions deal with generally accepted practices which have been followed in a particular business or industry over a period of time.

Domain-Level Aspects Vs Technical & Service-Level Aspects

Flexible Vocabulary: Technical aspects like security and service-level aspects such as pricing can have a generic vocabulary which could standardize over a period of time or through standard ontologies. The vocabulary of technical or service-level aspects remains similar across businesses or industries. However, in the case of domain-level aspects, though business rules and compliance are broad crosscutting concerns, their specific vocabulary vary. Consider the example of the Shipping industry – a business like FedEx® in the shipping business, has to comply with the Bioterorrism Act 2002 - Prior Notice for food shipments. Meanwhile, the aviation industry would have to comply with Federal Aviation Act. We note that though compliance remains an aspect across industries, the vocabulary for compliance is flexible. Due to its flexible vocabulary, domain-level aspects have to be specifically defined for each business by regulatory and governance (domain) experts. We define vocabulary for Compliance to Bioterrorism Act 2002 - Prior Notice (table 5.4) regulation in the Shipping industry. The regulation requires the consumer of the ShippingService ShipItem $^{(OP)}$ operation to intimate the FDA (Federal Drug Administration) with a prior notice for food shipments and use the prior notice confirmation while using ShipItem $^{(OP)}$.

Restricted Quantification: Quantification deals with the selection of services and other policy subjects in the services portfolio for applying a Policy i.e. it determines the policy scope. Unlike, technical and service-level aspects, the domain-level aspects have a limited quantification i.e. they do not have a broad impact on services in the services portfolio. Due to the nature of domain-level aspects they apply to specific services e.g. Compliance to Bioterrorism Act 2002 - Prior Notice applied to ShippingService ShipItem $^{(OP)}$ operation. Most importantly, the aspects and their vocabulary we identified in this section are more indicative than prescriptive. They could evolve as per specific business or industry. New aspects could derive from the existing aspects, or they could be entirely independent.

Name of Concern	Compliance to Bioterrorism Act 2002 – Prior Notice Regulation	
Type of Aspect	Bioterorrism Act 2002 - Prior Notice Compliance	
Related Aspects	Regulatory Compliance	
Context	Compliance notice to the service consumer while shipping food exports.	
Rationale & Discussion	The aspect deals with compliance to the Bioterrorism Act 2002 – Prior Notice which requires the consumer to use a prior notice confirmation number to ship food exports.	
Quantification	SHIPPINGSERVICE ShipItem$^{(OP)}$ Operation	
Vocabulary		
Vocabulary Terms	Type	Applicable Values
IsPriorNoticeRequired	Boolean	
PriorNoticeConfirmationNumber	String	

Table 5.4: Domain-Level Aspect – Compliance to Bioterrorism Act 2002 (Prior Notice)

5.4 Modeling the SHIPPINGSERVICE Scenario

We use a fictitious SHIPPINGSERVICE offered by FedEx® to explain the use of our service policy view. The service represents an underlying capability of shipping an item from one place to another. The SHIPPINGSERVICE defines a ShipItem$^{(OP)}$ operation which supports shipping a package. In addition to this, there could be other operations (fig 5.8) such as Get Rates and Transit Times$^{(OP)}$ – an operation which provides rates and transit times between two locations and Schedule Pick-up$^{(OP)}$ – an operation which supports pick-up of items from consumer's location. In addition, FedEx® also offers the TRACKINGSERVICE which supports tracking a shipment through its Track Shipment$^{(OP)}$ operation.

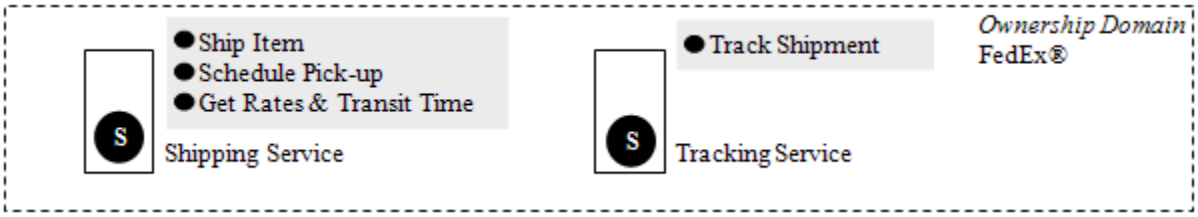


Fig 5.8: FedEx® Ownership Domain and the Services

The service capability view (fig 5.9) provides a functional view of the ShippingService. By applying a WSDL 2.0 transformation (from Appendix I) on the service capability view, the abstract service description could be obtained (fig 5.10).

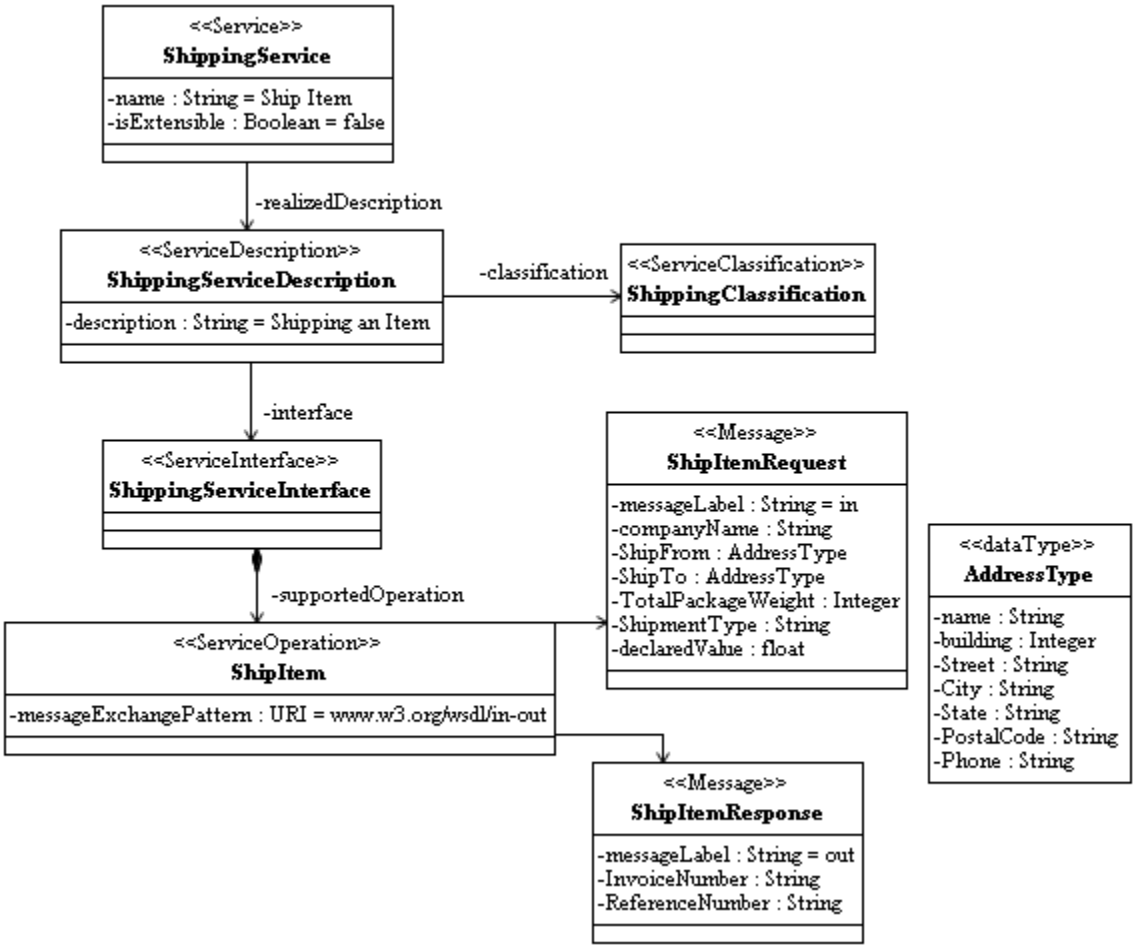


Fig 5.9: SHIPPINGSERVICE – *Service Capability View*

Below, we define an example pricing policy which is later applied to the ShipItem$^{(OP)}$ service operation. The domain experts use the service policy metamodel to independently define the pricing policy and associate it with the ShipItem$^{(OP)}$ policy subject. Defining the pricing policy inolves firstly defining the pricing domain vocabulary, secondly defining the alternatives by constraining the pricing vocabulary items and finally applying the policy on the ShipItem$^{(OP)}$ operation.

Pricing Policy: *Every customer could be a subscription customer or a regular customer. Subscription customers pay a propotional price based on their use. Regular customers pay an absolute price per service access.*

```
- <description xmlns="http://www.w3.org/ns/wsdl" xmlns:tns="http://service.fictitious.com/shipping/fedex"
    targetNamespace="http://service.fictitious.com/shipping/fedex">
    <documentation>Fictitious FedEx Shipping Service</documentation>
  - <types>
    - <xsd:schema xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        elementFormDefault="qualified" targetNamespace="http://service.fictitious.com/shipping/fedex">
      - <xsd:element name="AddressType">
        + <xsd:complexType>
        </xsd:element>
      - <xsd:element name="ShipItemRequest">
        - <xsd:complexType>
          - <xsd:sequence>
              <xsd:element minOccurs="1" maxOccurs="1" name="CompanyName" type="xsd:string" />
              <xsd:element minOccurs="1" maxOccurs="1" name="ShipFrom" type="tns:AddressType" />
              <xsd:element minOccurs="1" maxOccurs="1" name="ShipTo" type="tns:AddressType" />
              <xsd:element minOccurs="1" maxOccurs="1" name="TotalPackageWeight" type="xsd:integer" />
              <xsd:element minOccurs="1" maxOccurs="1" name="ShipmentType" type="xsd:string" />
              <xsd:element minOccurs="1" maxOccurs="1" name="DeclaredValue" type="xsd:float" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      - <xsd:element name="ShipItemResponse">
        - <xsd:complexType>
          - <xsd:sequence>
              <xsd:element minOccurs="0" maxOccurs="1" name="InvoiceNumber" type="xsd:string" />
              <xsd:element minOccurs="0" maxOccurs="1" name="Reference Number" type="xsd:string" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:schema>
    </types>
  - <interface name="ShippingServiceInterface">
    - <operation name="ShipItem" pattern="http://www.w3.org/ns/wsdl/in-out">
        <documentation>Ship Item Operation</documentation>
        <input element="tns:ShipItemRequest" />
        <output element="tns:ShipItemResponse" />
      </operation>
    </interface>
  + <binding xmlns:wsoap="http://www.w3.org/ns/wsdl/soap" name="ShippingServiceEndpoint"
      interface="tns:ShippingServiceInterface" type="http://www.w3.org/ns/wsdl/soap" wsoap:version="1.2"
      wsoap:protocol="http://www.w3.org/2006/01/soap11/bindings/HTTP/">
  + <service name="ShippingService" interface="tns:ShippingServiceInterface">
  </description>
```

Fig 5.10: WSDL 2.0 Snippet for abstract definition of the SHIPPINGSERVICE

Step I: Defining the Service Pricing Policy Domain

The first step for the pricing expert (domain expert) is to define the domain vocabulary for the service pricing domain (fig 5.11) if it is not done previously. The domain vocabulary is defined using visual models using the Service Policy View.

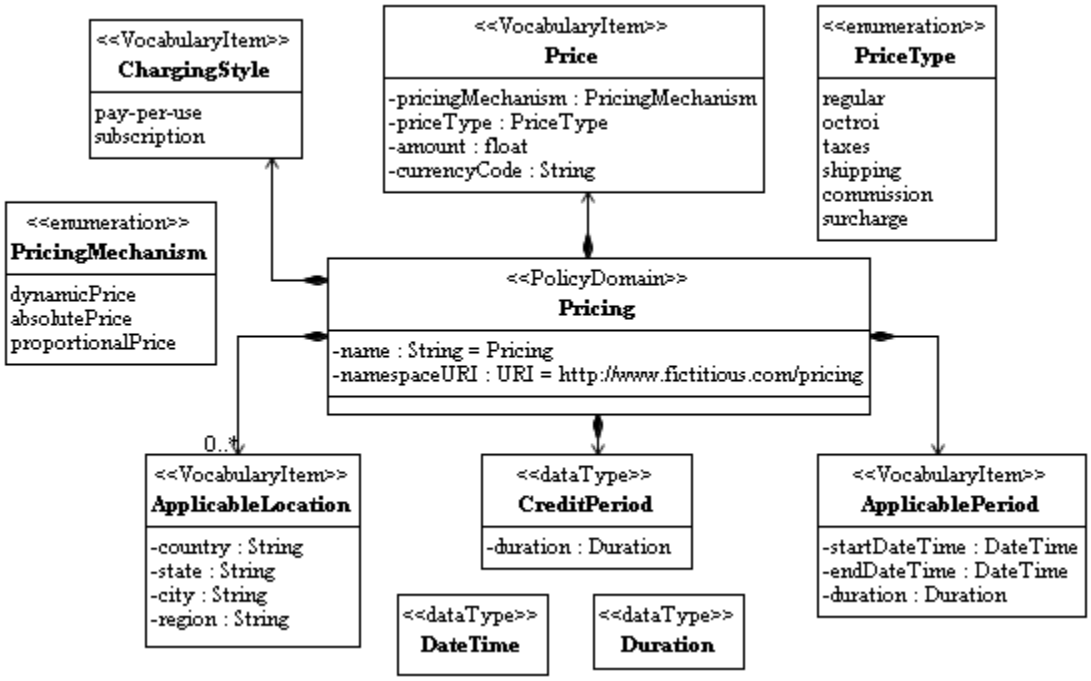


Fig 5.11: Vocabulary Definition – Pricing

Step II: Defining the Pricing Policy

After modeling the pricing vocabulary, the pricing policy has to be modeled by the domain experts. Fig 5.12, shows the pricing policy modeled using our services policy metamodel.

Step III: Quantification – Applying the Pricing Policy to ShipItem$^{(OP)}$

Once the pricing policy id modeled, it has to be applied to the ShipItem$^{(OP)}$ operation (fig 5.13) – a policy subject.

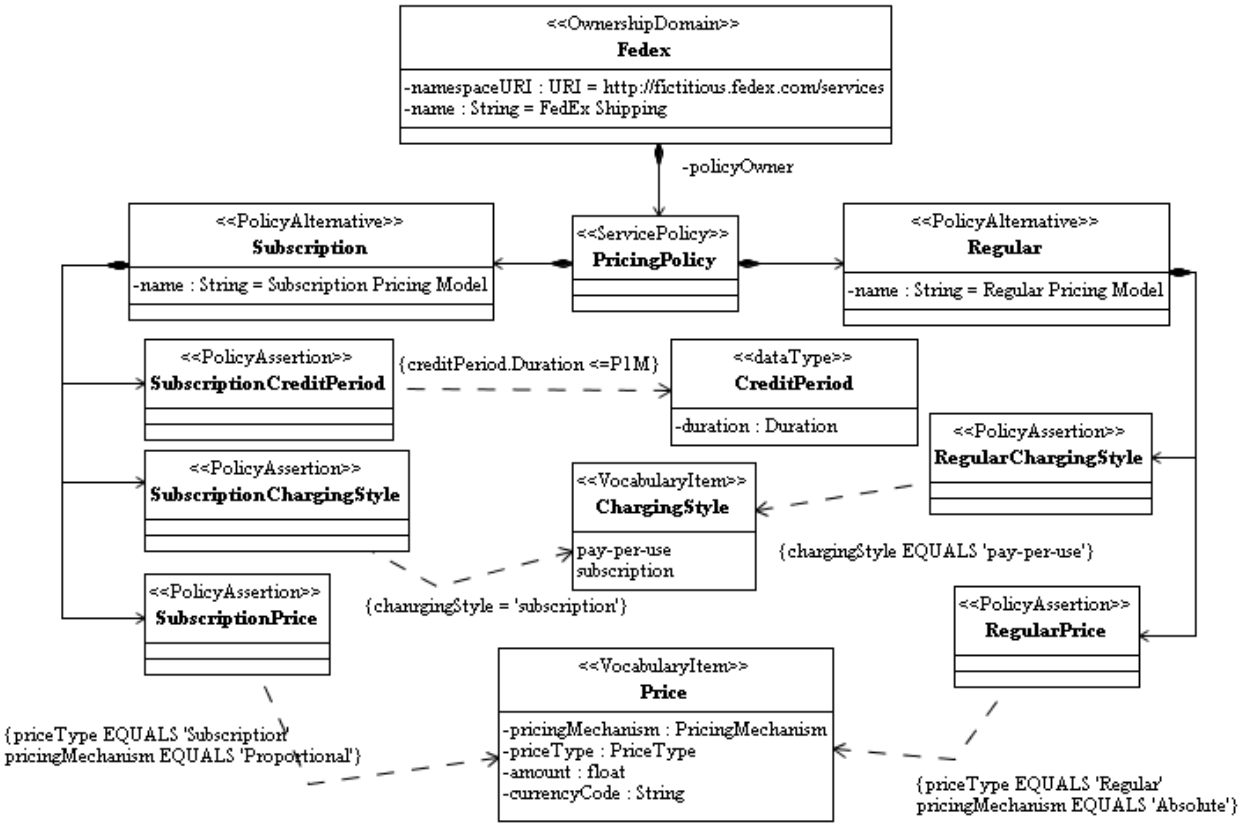


Fig 5.12: Service Pricing Policy Model

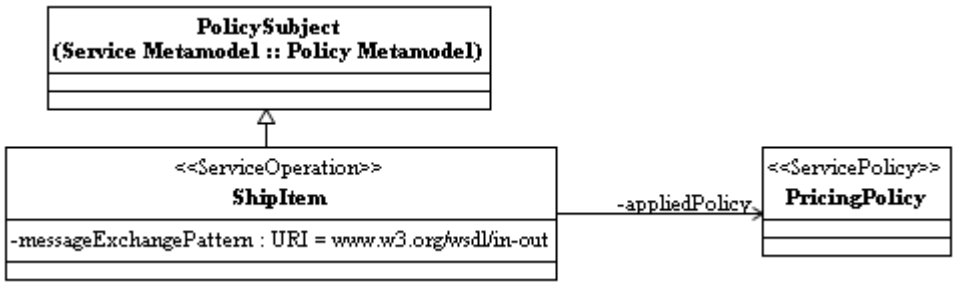


Fig 5.13: Service Pricing Policy Model

5.5 Models to Executable Specifications

Once the domain experts model the policies using our service policy view, these policy models have to be converted to appropriate interoperable standards. The policies should also be incorporated into service descriptions.

For the SHIPPINGSERVICE, the service capability model (in fig 5.9) captured the underlying capability on offer. The capability model was then converted to a standard WSDL 2.0 service description (in fig 5.10). In a similar manner, the policies described using our service policy models have to be transformed to appropriate industry accepted interoperable standards. Since there are multiple and sometimes competing standards, we look at different standards available in each of layers of the generic policy model (table 5.5).

Generic Policy Model Layer	Specifications
Vocabulary Specification	- XML Schema [126] - Web Ontology Language (OWL) [127] to support specification of domain ontologies
Constraint Specification	- Domain Dependent Specification: Domain specific assertions using WS-SecurityPolicy, WS-Trust, WS-ReliableMessagingPolicy [128]. - Domain Independent Specification: Domain independent assertions using WS-PolicyConstraints [129], XACML [130].
Policy Specification	- WS-Policy - Web Service Policy Language (WSPL) [131]
Binding Specification	- WS-PolicyAttachment

Table 5.5: Standards Relevant to Generic Policy Model Layers

Technically, the policy models created using our service policy metamodel could be transformed to any of these specifications using MOF2 Models to Text Transformation Language (MTL) standard mappings (or any other mapping language). But we have made certain choices about the standards we would use for our standard transformations. These choices are based on two considerations – current industry adoption and support for generic processing of policies.

Based on industry adoption we choose WS-Policy specification to specify policies. WS-Policy specification has a solid industry backing and is a mature W3C recommendation now. SOA vendors also support policies defined using WS-Policy in their middleware software. Having chosen WS-Policy, choosing WS-Policy Attachment was an obvious option for binding specification. For vocabulary specification and constraint specification: Domain-dependent constraint specification languages like WS-Security policy (security domain) and WS-ReliableMessagingPolicy (reliable messaging domain) have matured and evolved with WS-Policy. They provide standard semantics and constraints to specify security and reliable messaging capability. However, we choose a domain-independent constraint specification language – WS-PolicyConstraints. WS-PolicyConstraints help to specify domain-independent generic constraints using XACML-based functions. We choose the nascent WS-PolicyConstraints over the much adopted domain-dependent constraint languages for the following reason:

- Absence of existing assertion languages to specify domain-specific assertions for service-level aspects such as availability, pricing, promotions as well as domain-specific aspects.
- To provide flexibility in rich vocabulary specification for service-level and domain-level aspects across industries and businesses. Domain-dependent assertion languages have currently restricted vocabulary to improve interoperability.
- Advantage of using a common generic policy handling logic for parsing policies in the SOA middleware instead of having multiple policy handlers.

We have developed a MTL transformation to transform the model developed using the service policy metamodel to preferred specifications (XML Schema, WS-PolicyConstraints, WS-Policy and WS-PolicyAttachment) (fig. 5.14). The Normal Form of WS-Policy is chosen for the transformation.

5.5.1 Transforming the Pricing Policy Model

```
// Model to Text Language Transformation
// Service Policy Metamodel to WS-Policy Normal Form
@text-explicit
[template public ModelToWSPolicySpec (policy : ServicePolicy) ]

<wsp:Policy
   xmlns:wsp="http://www.w3.org/ns/ws-policy"
   xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
   xmlns:wspc= "http://research.sun.com/ns/ws-policyconstraints"
   xmlns:wsu="http://docs.oasis-open.org/wssecurity-utility-1.0.xsd"
   wsu:Id = '[policy.name]'>

   <wsp:ExactlyOne>

      [For (alternative:PolicyAlternative | policy.alternatives) ]
         <wsp:All>
            [For (assertion:PolicyAssertion | alternative.assertions) ]
                [AssertionToWSPolicyConstraint(assertion) /]
            [/For]
         </wsp:All>
      [/For]
   <wsp:ExactlyOne>
<wsp:Policy>
[/template]


//Transform Assertion to WS-PolicyConstraint
@text-explicit
[template public AssertionToWSPolicyConstraint(assertion: PolicyAssertion)]
   <wspc:Apply FunctionId = "&function; [assertion.operator]">
      <wspc:ResourceAttributeDesignator
         AttributeId = "[assertion.constrainedElement.name]"
         DataType    = "[assertion.constrainedElement.type]"  />
      <wspc:AttributeValue DataType = &type; [assertion.specification.type]>
         [assertion.specification.stringValue()]
      </wspc:AttributeValue>
   </wspc:Apply>
[/template]
```

Fig 5.14: MTL Transformation (Service Policy Metamodel to Specifications)

We apply our standard MTL transformation to the pricing policy model. The pricing vocabulary we defined is converted to XML Schema using standard XML mappings (fig 5.15).

```
<?xml version="1.0" encoding="utf-8" ?>
- <xs:schema xmlns="http://www.fictitious.com/servicepricing" elementFormDefault="qualified"
    targetNamespace="http://www.fictitious.com/servicepricing"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include schemaLocation="Common.xsd" />
  - <xs:element name="Pricing">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element xmlns:q1="www.fictitious.com" name="ApplicablePeriod"
            type="q1:Validity" />
          <xs:element xmlns:q2="www.fictitious.com" name="ApplicableLocation"
            type="q2:Location" />
          <xs:element name="CreditPeriod" type="xs:duration" />
        - <xs:element xmlns:q3="www.fictitious.com" name="Price">
          - <xs:complexType>
            - <xs:sequence>
              - <xs:element name="PricingMechanism">
                + <xs:simpleType>
                </xs:element>
              - <xs:element name="PriceType">
                + <xs:simpleType>
                </xs:element>
                <xs:element name="Amount" type="xs:float" />
                <xs:element name="CurrencyCode" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        - <xs:element name="ChargingStyle">
          - <xs:simpleType>
            - <xs:restriction base="xs:string">
                <xs:enumeration value="pay-per-use" />
                <xs:enumeration value="subscription" />
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

Fig 5.15: Pricing Vocabulary XML Schema

The pricing model is transformed to WS-Policy with the policy constraints being expressed in WS-PolicyConstraints (fig 5.16) and the pricing policy is attached to the ShipItem $^{(OP)}$ using WS-PolicyAttachment (fig 5.17). A pricing policy handler (a Policy Enforcement Handler) is optionally generated and added in the SOA middleware to handle pricing policies (more on this in the next section).

```
<wsp:Policy wsu:Id = "ServicePricingPolicy">
    <wsp:ExactlyOne>
      <!-- Policy Alternative: Subscription Customers -->
      <wsp:All>
         <Apply FunctionId="&wspc;function:is-less-than-or-equal">
            <AttributeValue DataType="&xsd;duration">P1M</AttributeValue>
            <ResourceAttributeDesignator AttributeId="creditPeriod" DataType="&xsd;duration"/>
         </Apply>

         <Apply FunctionId="&wspc;function:equals">
            <AttributeValue DataType="&xsd;string">Regular</AttributeValue>
            <ResourceAttributeDesignator AttributeId="price/priceType" DataType="&xsd;string"/>
         </Apply>

         <Apply FunctionId="&wspc;function:equals">
            <AttributeValue DataType="&xsd;string">proportional</AttributeValue>
            <ResourceAttributeDesignator AttributeId="price/pricingMechanism" DataType="&xsd;string"/>
         </Apply>

         <Apply FunctionId="&wspc;function:equals">
            <AttributeValue DataType="&xsd;string">subscription</AttributeValue>
            <ResourceAttributeDesignator AttributeId="chargingStyle" DataType="&xsd;string"/>
         </Apply>
      </wsp:All>
      <wsp:All>
      <!-- PolicyAlternative: Regular Customers -->
         <Apply FunctionId="&wspc;function:equals">
            <AttributeValue DataType="&xsd;string">Regular</AttributeValue>
            <ResourceAttributeDesignator AttributeId="price/priceType" DataType="&xsd;string"/>
         </Apply>
         <Apply FunctionId="&wspc;function:equals">
            <AttributeValue DataType="&xsd;string">absolute</AttributeValue>
            <ResourceAttributeDesignator AttributeId="price/pricingMechanism" DataType="&xsd;string"/>
         </Apply>
         <Apply FunctionId="&wspc;function:equals">
            <AttributeValue DataType="&xsd;string">pay-per-use</AttributeValue>
            <ResourceAttributeDesignator AttributeId="chargingStyle" DataType="&xsd;string"/>
         </Apply>
      </wsp:All>
        </wsp:ExactlyOne>
  </wsp:Policy>
```

Fig 5.16: Pricing Policy Definition in WS-Policy and WS-PolicyConstraints

```
<wsp:PolicyAttachment>
   <wsp:AppliesTo>
      <wsa:URI>http://...#interface/operation(ShippingServiceInterface/ShipItem)</wsa:Address>
   </wsp:AppliesTo>
   <wsp:PolicyReference URI="http://www.fictitious.com/policies#ServicePricingPolicy" />
</wsp:PolicyAttachment>
```

Fig 5.17: External Policy Attachment using WS-PolicyAttachment

5.6 Policy Enforcement at the SOA Middleware

Once the policies are modeled and associated to the policy subjects, the service descriptions are enhanced with policy information. Now these policies have to be enforced in the SOA middleware. The most important criterion for policy enforcement is that it has to be unintrusive. We use an active SOAP intermediary – the Policy Enforcement Point intermediary (PEP) Intermediary (fig 5.18). The PEP intermediary works on the SOAP headers associated with service policies. We use Apache Axis 2.0 [132] (herein Axis2) SOAP engine as our PEP SOAP intermediary. We take advantage of the extensible SOAP processing model of Axis2.

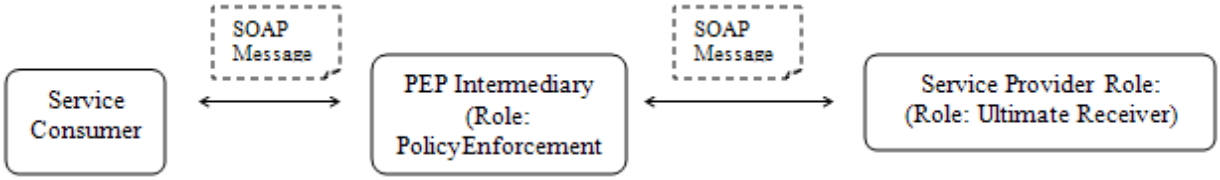


Fig 5.18: Unintrusive Policy Enforcement using PEP Intermediary

Inside the PEP intermediary (fig 5.19), we have a Generic Handler (an XACML policy processor) to handle all service policies – an advantage of using the domain-independent WS-PolicyConstraints. However, if we need application specific programming logic to handle special policy enforcement for certain policy domains, we could optionally choose to have an exclusive Policy Enforcement Handler (PE Handler). The generic handler and the optional PE handlers are part of a user-defined Policy Enforcement Phase (PE Phase). As soon as a new instance of a policy domain is added in the policy model, a corresponding policy enforcement handler is optionally generated and automatically added to the end of the PE phase. The SOAP headers representing different aspects such as security, pricing etc. have the role http://fictitious.com/role/policyEnforcement and the PE intermediary which plays the policy enforcement role must understand and process these headers.

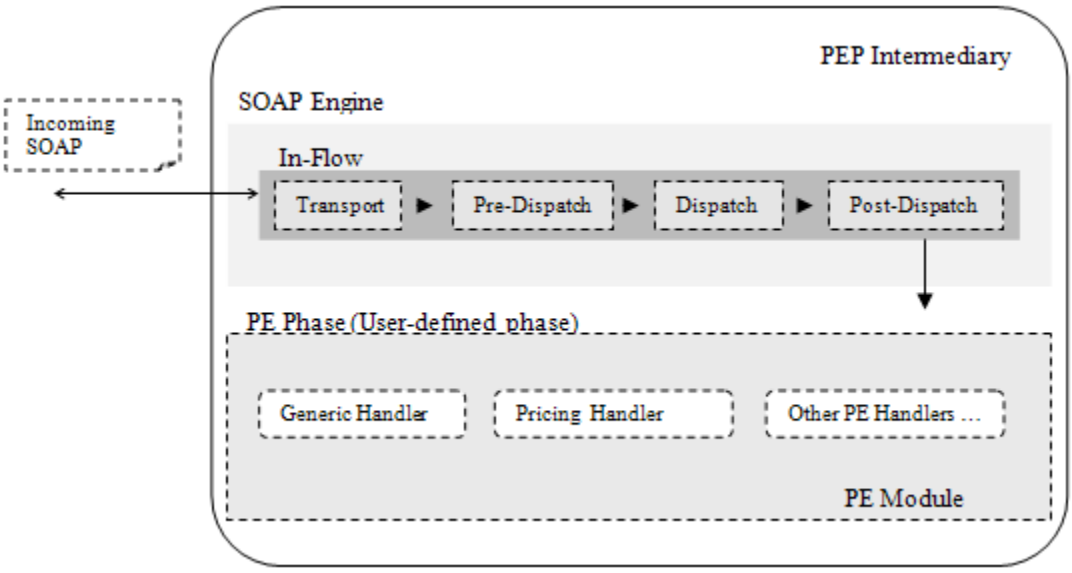


Fig 5.19: Inside the PEP Intermediary

```
- <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  - <soap:Header>
    - <s:Security xmlns:s="http://fictitious.com/security"
        soap:role="http://fictitious.com/role/policyEnforcement" soap:mustUnderstand="true">
        <s:Username>eBayUser_10312</s:Username>
        <s:PasswordType>Clear Text</s:PasswordType>
        <s:PasswordValue>eBay@!456</s:PasswordValue>
      </s:Security>
    - <p:Pricing xmlns:p="http://fictitious.com/servicepricing"
        soap:role="http://fictitious.com/role/policyEnforcement" soap:mustUnderstand="true">
      - <p:Price>
          <p:priceType>regular</p:priceType>
        </p:Price>
        <p:chargingStyle>subscription</p:chargingStyle>
      </p:Pricing>
    </soap:Header>
    <soap:Body>..</soap:Body>
  </soap:Envelope>
```

Fig 5.20: Sample SOAP Request for the ShipItem$^{(OP)}$ Operation

Once the policy enforcement is done, the SOAP messages are routed to the ultimate receiver (or the service provider). Fig 5.20 presents a sample SOAP request for the ShipItem $^{(OP)}$ operation (the SOAP body is not presented for brevity). The header element Service Pricing would be interpreted by the PEP intermediary based on the role. The message is then handled by Pricing Handler.

5.7 Related Work

Model-driven approaches to developing webservices [104-106] are increasingly getting popular. OMG realized the need to standardize model-driven services development – the result – RFP (request for proposal) UML Profile and Metamodel for Services (UPMS) [133], hereon RFP UPMS. However, the RFP UPMS does not address Service Policies; the focus is on Services Modeling – capability and contract modeling. The OASIS SOA Reference Model (SOA-RM) [32] and the WS-Arch [94] (Webservices Architecture) describe service policies in detail. Our approach complies with the SOA-RM. In our approach, we consider all aspects of service policy modeling by addressing the 4-layers of the generic policy model.

A close related work – Ortiz et al.'s [134] work on modeling extra-functional properties deals with modeling services based on the Service-Component Architecture (SCA) and defining extra-functional properties. They have developed a UML Profile for SCA and to model extra-functional properties [135]. However the focus of their approach is not on independent policy development – by describing alternatives and constraints – instead the focus is on defining extra-functional properties at the modeling level and representing it using WS-Policy. Policy enforcement implementations are based on aspect-oriented techniques [136]. Moreover, the aspects dealt (e.g. logging) are more technical in nature; in comparison our approach addresses technical, service-level and domain-level aspects. Also Ortiz et al.'s approach does not address vocabulary specification for policy domains and constraint specification.

With respect to vocabulary specification, O'Sullivan has done extensive work on non-functional properties in service descriptions; he has also produced concrete XML syntax of service properties [137], which could be reused as vocabularies. Also ontologies such as QoSOnt [138] (an ontology for QoS) could be reused to describe policy vocabulary.

With respect to policy enforcement implementations, we use a SOAP intermediary to handle policy enforcements. However, there are variety of approaches [139, 140] (including Ortiz et al.'s) using aspect-oriented programming techniques to handle crosscutting aspects like service management and adaptability. Our approach could complement those approaches and provide means to identify aspects and aid in the entire

life-cycle of service policy development. Later, we could enhance our approach to support AOP-based quantification.

5.8 Summary

In this chapter we addressed broad-based independent service policy development using our service policy view and the policy metamodel. We addressed service policy development in the early-stage services development based on the generic policy framework. We also addressed different levels of policy aspects – technical, service-level and domain-level aspects and vocabulary associated with these aspects. We presented an aspect catalog to define aspects related to policy domains. We demonstrated our policy modeling approach using the SHIPPINGSERVICE and a service pricing policy example. We also explained policy enforcement using the PEP intermediary.

Chapter 6

Service Flavors Strategy

Differentiating Services in a Services Marketplace

In a services marketplace where a service is provided by multiple service providers, service offerings have to be differentiated against competitor services. Differentiation helps to sustain as well as grow market share. Strategies to differentiate service offerings have to be *unintrusive* – without requiring major changes to the existing service realization mechanisms. In order to unintrusively differentiate services, we need a differentiation strategy – a method to identify and document differentiating aspects of a service and manipulate them to differentiate services from that of competition.

6.1 Need for Service Differentiation

In the context of a services marketplace, a service can be a *commoditized service*, a *specialized service* or a *monopolistic service* based on the number of service providers providing that service. Monopolized services are provided by a single service provider (e.g. eGovernance services provided by the Government). Specialized services are provided by very few service providers in the services marketplace (e.g. payroll & benefits services). On the contrary, commoditized services are always provided by multiple competing service providers in a services marketplace. For example, consider the SHIPPINGSERVICE in the context of an e-marketplace such as eBay® provided by multiple providers such as UPS®, USPS®, DHL®, OverniteExpress® and FedEx®. More often than not, the underlying capabilities represented by competing commoditized services remain the same. Additionally, competing services may even have standardized (open standards-compliant) messages and interfaces. Standardization leads to business layer interoperability, efforts

such as Universal Business Language (UBL) [141], ebXML [142], RosettaNet [143] and UN/CEFACT address business layer interoperability [144]. The standardization of these competing services is a result of market compulsions or regulatory compliance requirements (in case of HL7 [145] and SWIFT [146]). For customers, standardization supports easy migration from one provider to another. However, standardization takes away provider lock-in advantages for service providers. As a result, every service provider is faced with the dilemma of balancing standardization and differentiation of their service offerings. Given that standardization is a necessity, service providers of commoditized services are pressed with the challenge to differentiate their service offerings from that of the competition in order to sustain as well as grow market share. However, in case of the monopolistic and specialized services, the need to differentiate is not as much as the need to differentiate commoditized services. Differentiation of commoditized services is possible by providing competitive and differentiated offerings [147,148]. A service development and delivery platform must support service providers in creating both competitive and targeted offerings of their services.

Competitive Offerings: Given the functionality of the services are the same (due to market enforced standardization), differentiation of commoditized services is often done through competitive pricing, promotions, enhancing the reach of service offerings and improving quality of service. By understanding the differentiating aspects among competing services and manipulating these aspects competitively we can create competitive offerings. Ideally, it is done by making the terms of offer of a service attractive for a consumer.

Targeted Offerings: Creating discrete variations of services specially targeted towards a market segment or a customer niche. These discrete service variations –*Service Flavors*, as we call it – are created through customizing differentiating aspects and making the service attractive to the consumer.

6.2 Flavoring Aspects: Differentiating Aspects of a Service

Before attempting to differentiate a commodity service, it is important to understand the *changing parts* in a service description across service offerings. By understanding the changing parts that influence a consumer's decision to choose a service offering, we could identify differentiating aspects of a service that help in differentiating it against competing services. As we discussed earlier, every service represents an underlying *capability on-offer* offered under specific terms and conditions – the *terms of offer*. The capability on-offer satisfies the goal of a consumer under the constraints of the terms of offer. Terms of offer – aspects such as cost, discounts, availability, quality of service, convenience of use, and packaging – could be represented as service properties. Most often, the capability on-offer represents the functional aspects, whereas the terms of offer represent the non-functional aspects of a service. A service description must describe both the capability on-offer as well as the terms of offer for automatic selection and consumption of a service.

In case of our commodity SHIPPINGSERVICE, the underlying capability is to ship items from one place to another. Given that the capability on-offer is the same across competing services in a services marketplace, how does a service provider differentiate his shipping service from that of the competition? On what basis does a service consumer choose a particular shipping service? Consider the examples of a websites such as www.redroller.com and iShip™ that compare the shipping services provided by various providers such as USPS®, DHL® and OverniteExpress®. It is interesting to note the terms at which these services are compared – *delivery date* (quality of service) and *shipping rates* (price). Therefore, given that the capability on-offer is the same, consumers would choose a particular service based on *attractiveness* of the terms of offer. Certain aspects of the terms of offer which could make service offerings attractive to consumers would become *differentiating aspects*.

Competitive offerings could be created by appropriately varying these differentiating aspects and attractively positioning services to consumers. Targeted offerings could be created by offering the same capability on-offer under different terms of offer (fig 6.1). Each targeted offering represents a discrete variation of a service or a *Service Flavor*. Since

the differentiating aspects are used in service flavoring, they are also called *Flavoring Aspects*. Some of our domain-level, technical or service-level aspects, we identified in the previous chapter could be flavoring aspects. The important criteria to determine if an aspect is a flavoring aspect or not is to answer this question – 'would the aspect make the terms of offer attractive to the consumer and differentiate the service offering from that of competition?' We call this the *attractiveness of terms of offer criteria*. There are other aspects such as service reputation, market perception and service rating by rating agencies which also significantly impact the choice of a service by a consumer. However, we do not address them as they are not under the direct and explicit influence of the service provider – though they are implicitly addresses by other aspects.

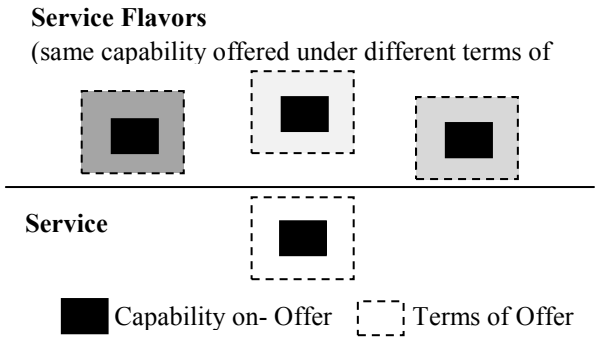


Fig 6.1: Service Flavors – Targeted Offerings

It is important to identify differentiating aspects during early-stage design of services and provide a way to alter them appropriately and unintrusively to support differentiation of services. Some of the non-functional aspects we identified in the Chapter 5 could ideally be differentiating flavoring aspects – i.e. they would satisfy the *attractiveness of terms of offer criteria*. A case in point is the service pricing and promotions aspects. By attractively pricing and promoting service offerings, service providers can effectively differentiate their service offerings thereby gaining market share. In this chapter, we use service pricing and service promotion as examples of flavoring aspects to explain our flavoring strategy. We documented the service-level service pricing flavoring aspect in the previous chapter using our standard schema. Similarly, we also document service promotion in table 6.1.

Name of Concern	Service Promotion	
Type of Aspect	Promotion	
Related Aspects	Discounts & Rewards	
Context	A Service provider would promote his service by offering discounts and rewards to service consumers.	
Rationale & Discussion	Promotion is a concern which deals with promoting the use of a service offering among service consumers. Promotion schemes could provide discounts on using the service, waive cost for a fixed time-period or offer reward points which could be redeemed.	
Vocabulary		
Vocabulary Terms	Type	Applicable Values
Promotion Period	Validity	
Applicable Location	Location	
Reward Type	String	Reward Points, Coupons
Reward Value	Integer	
Discount Percent	Integer	
Discount Value	Float	

Table 6.1: Service Promotion – A Flavoring Service-Level Aspect

We model the service promotion vocabulary (fig 6.2) and we also show the corresponding XML schema generated using the standard XML mapping (fig 6.3).

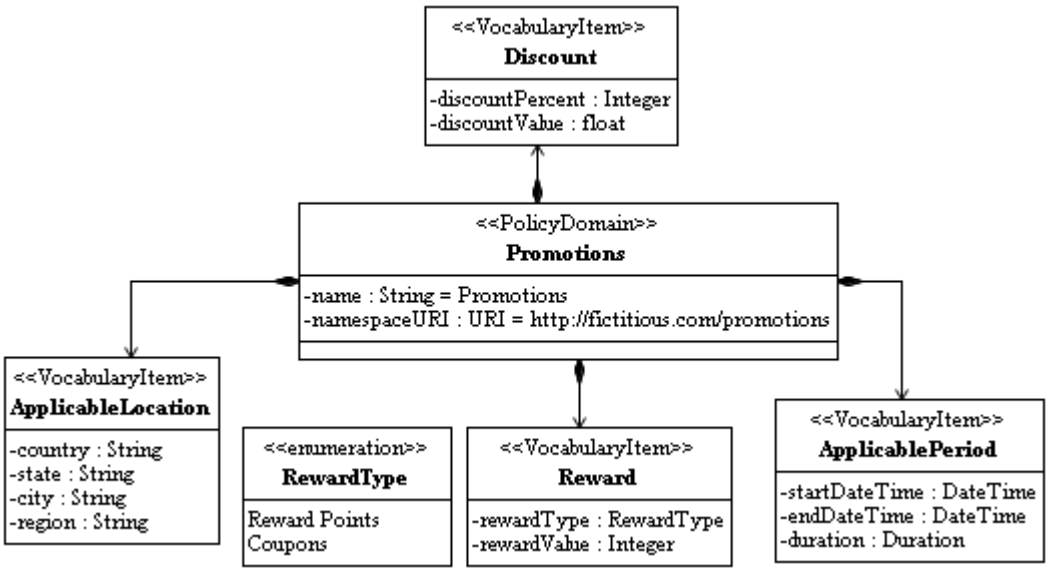


Fig 6.2: Vocabulary Definition – Promotions

6.2.1Note on Vocabulary Items for Flavoring Aspects

The vocabulary items associated with the flavoring aspects have to standardize across service providers offering a particular service in a specific domain. The standardization leads to accurate comparison of the terms of offer across service offerings. The standardization of vocabulary in an industry or domain could happen through consensus or evolution. For example the *shippingCost* and *deliveryTime* are now standardized vocabulary items in the package shipping industry. Another efficient way of achieving consensus on vocabulary in a particular domain could be through domain ontologies [149]. For example, QoSOnt [138] – ontology for quality of service for service-centric systems could be used as a vocabulary for the service quality aspect. Vocabulary items are different in different industries and businesses. For example, promotions in the shipping & logistics domain could be based on *reward points*, but in the aviation domain, promotions are based on accumulated *flyer miles*. For this reason, the vocabulary items we defined for service pricing and promotion are more prescriptive than indicative and hence could be extended or replaced completely.

```
<?xml version="1.0" encoding="utf-8" ?>
- <xs:schema xmlns="www.fictitious.org" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include schemaLocation="Common.xsd" />
  - <xs:element name="ServicePromotion">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element name="PromotionPeriod" type="Validity" />
          <xs:element name="ApplicableLocation" type="Location" />
        - <xs:element minOccurs="0" maxOccurs="unbounded" name="Reward">
          - <xs:complexType>
            - <xs:sequence>
              - <xs:element name="RewardType">
                - <xs:simpleType>
                  - <xs:restriction base="xs:string">
                      <xs:enumeration value="RewardPoints" />
                      <xs:enumeration value="Coupons" />
                    </xs:restriction>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="RewardValue" type="xs:integer" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        - <xs:element minOccurs="0" maxOccurs="unbounded" name="Discount">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="DiscountPercent" type="xs:positiveInteger" />
                <xs:element name="DiscountValue" type="xs:float" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Fig: 6.3 Promotions Vocabulary – XML Schema

6.3 Differentiating Services with Service Flavors

Consider the example of a targeted offering of the SHIPPINGSERVICE - **Subscription Service Flavor** targeting a customer niche, subscription customers. The subscription service flavor supports the business strategy of attracting more subscription customers in order to have predictability in revenues. The strategy to attract subscription customers is by offering them a one-month credit period. The subscription service flavor adheres to the following Subscription Pricing policy:

"The service provider of the fictitious ShippingService provides a one-month credit period for its subscription customers."

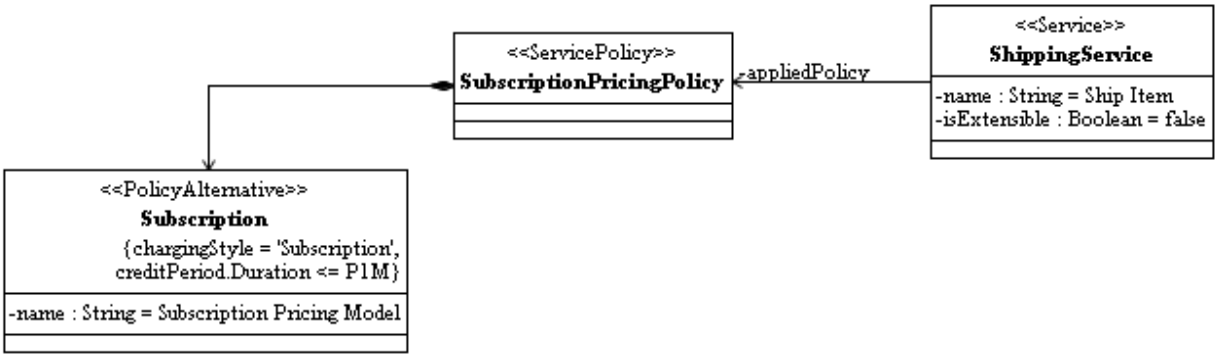


Fig 6.4: Subscription Pricing Policy

On using our standard transformation, the *creditPeriod* and the *chargingStyle* vocabulary items of the pricing policy domain is constrained. Fig 6.5 provides a WS-PolicyConstraint representation of the *creditPeriod*. In fig 6.6, the SHIPPINGSERVICE WSDL2.0 description is enhanced with the subscription pricing policy.

```
<Apply
   FunctionId="&wspc;function:is-less-than-or-equal">
<AttributeValue
   DataType="&xsd;duration">P1M</AttributeValue>
<ResourceAttributeDesignator
   AttributeId="creditPeriod"
   DataType="&xsd;duration"/>
</Apply>
```

Fig 6.5: WS-PolicyConstraints on creditPeriod vocabulary item

The subscription service flavor is created by offering the SHIPPINGSERVICE with different terms of offer for the subscription customers. The subscription pricing policy is specified with a single alternative which has domain-independent assertions on vocabulary item *creditPeriod* and *chargingStyle* specified using WS-PolicyConstraints. The policy is intrinsically referenced in the <service/> using the WS-Policy Attachment's <wsp: Policy Reference />. The enhanced service description describes the targeted offering - Subscription Service Flavor for the subscription customers (market segment).

```
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://service.fictitious.com/shipping/fedex"
             xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:wsp="http://www.w3.org/ns/ws-policy"
             xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
             xmlns:wspc= "http://research.sun.com/ns/ws-policyconstraints"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
             targetNamespace="http://service.fictitious.com/shipping/fedex">

    <documentation>Fictitious FedEx Shipping Service</documentation>
   <types> ....   </types>
   <wsp:Policy wsu:Id="SunscriptionPricingPolicy">
         <wsp:ExactlyOne>
               <wsp:All>
                     <Apply FunctionId="&wspc;function:is-greater-than-or-equal">
                     <AttributeValue DataType="&xsd;duration">P1M</AttributeValue>
                     <ResourceAttributeDesignator AttributeId="creditPeriod" DataType="&xsd;duration"/>
                     </Apply>

                     <Apply FunctionId="&wspc;function:equals">
                     <AttributeValue DataType="&xsd;string">Subscription</AttributeValue>
                     <ResourceAttributeDesignator AttributeId="chargingStyle" DataType="&xsd;string"/>
                     </Apply>
               </wsp:All>
         </wsp:ExactlyOne>
   </wsp:Policy>

   <interface name="ShippingServiceInterface"> ... </interface>
   <binding> ... </binding>

   <service name="ShippingService" interface="tns:ShippingServiceInterface">
      <documentation>Fictitious FedEx Shipping Web Services.</documentation>
      <endpoint name="ShippingServiceEndpoint" binding="tns:ShippingServiceEndpoint"
                address="http://service.fictitious.com/shipping/fedex.asmx"/>
      <wsp:PolicyReference URI="#SubscriptionPricingPolicy" />
   </service>
</description>
```

Fig 6.5: Subscription Service Flavor

6.3.1 Service Flavors – Customer Context-aware Policies

Segmenting markets and targeting those market segments with promotions is a good service differentiation strategy [147]. Service flavors have to be created for each of these market segments (e.g. subscription service flavor). For example, the pricing or promotions could be different for members of an alliance (e.g. members of Star Alliance in the aviation sector can redeem frequent *flyer miles* (reward points) across member airlines) from that of other consumers.

The service consumers could be segmented based on various customer segmentation schemes – based on customer characteristics such as small businesses, business partners, or members of an alliance or based on qualitative characteristics such as gold, silver or platinum customers derived based on previous engagements or revenues from the customer. The domain expert defines consumer segments and associates service policies

to that segment. For example, consider a Service Flavor: **USFSB Service Flavor** – a targeted offering strategy (fig 6.6):

"The promotion policy provides a flat 10% discount on the ShippingService to members of the United States Federation of Small Businesses (USFSB, Inc.). "

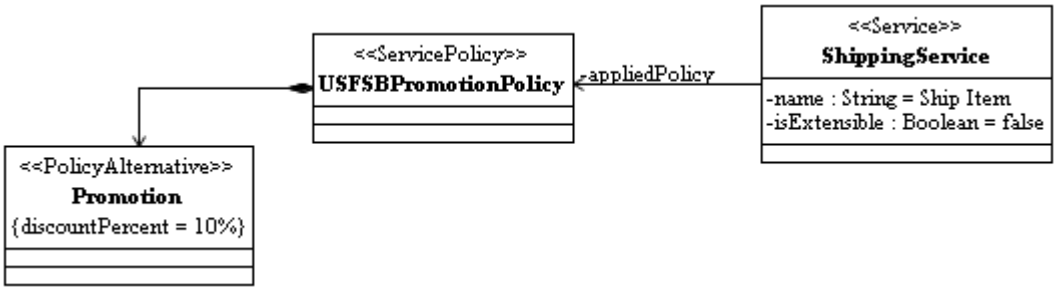


Fig 6.6: USFSB Service Flavor

At run-time, the service consumer's market segment is identified and the appropriate service flavor is provisioned for the consumer. In order to support this at the SOA middleware, we have a Consumer Profiling Handler (CPH) (fig 6.7) which is the first handler that gets invoked in the PE phase before the generic and the other PE handlers. The CPH deals with identifying and profiling the consumer. The consumer profile information is shared with the other handlers using the *MessageContext*.

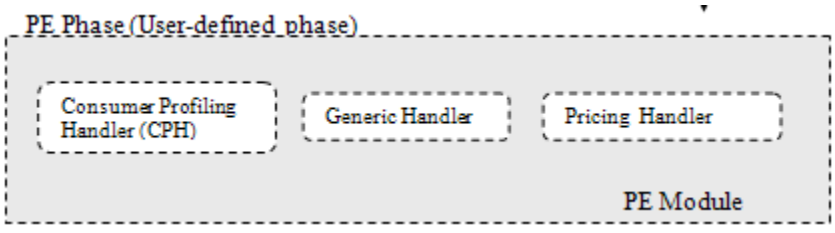


Fig 6.7: Consumer Profiling Handler

The SOAP header would have the consumer profile information (fig 6.8). It provides the consumer reference UUID (global unique identifier), the access time, location and the formatted name of the consumer.

```
- <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  - <soap:Header>
    - <consumer:ConsumerProfile xmlns:consumer="http://fictitious.com/consumerprofile"
        xmlns:common="http://fictitious.com/common"
        soap:role="http://fictitious.com/role/policyEnforcement"
        soap:mustUnderstand="true">
        <consumer:reference scheme="UUID">00300571-cecb-1dec-978d-
          559058888227</consumer:reference>
        <consumer:FormattedName>Take5 Inc</consumer:FormattedName>
        <common:accessTime>2008-03-29T13:20:00.000</common:accessTime>
      - <common:Location>
          <common:Country>US</common:Country>
        </common:Location>
      </consumer:ConsumerProfile>
    </soap:Header>
    <soap:Body>..</soap:Body>
  </soap:Envelope>
```

Fig 6.8: SOAP Request with Consumer Profile Information

6.4 Related Work

Service Flavors present a strategy to differentiate services in the service marketplace from the perspective of the provider. The closest related work is the Webservices Offering Language (WSOL) [150], but WSOL is not specifically intended for differentiating services. It is designed to support management of services, expressing constraints (pre-, post-conditions) etc. Though the 'class of service' concept in WSOL could be considered to create a service flavor, it is not standards compliant. The WSOL specification presents a WSDL1.1-compatible custom-XML language to describe a service offering. In comparison, Service Flavors is open-standards compliant (WS-Policy, WS-Policy Attachment and WS-PolicyConstraints). Service Flavors supports the entire life-cycle from early-stage identification, definition and documentation of differentiating (flavoring) aspects as well as the vocabulary items associated with them.

Flavors could be seen as services with different Service-Level Agreements (SLA's). For defining SLAs, there are languages such as the IBM's Web Service-Level Agreement (WSLA) [151] and HP's Webservice Management Language (WSML) [152]. These languages support QoS guarantees than really defining discrete variations. WS-QoS [153] is a QoS specification language which has a notion of class of service; however it is centered on network-level QoS and is not useful to flavor services. Flavoring aspects represent crosscutting service-level concerns from the perspective of the service provider.

Aspect-Oriented Software Development (AOSD) [154] offers an elegant way to handle cross-cutting concerns in software development by modularizing these concerns as aspects. The flavoring aspects represent crosscutting concerns such as availability, quality of service, pricing and promotions which are largely service-level provider concerns. There could be other crosscutting concerns in services development such as domain-level concerns, technical middleware concerns, service realization concerns (implementation and composition concerns) which are not addressed by flavoring aspects.

Certain flavoring aspects are non-functional in nature. Definition of non-functional properties of a service is addressed by frameworks such as OWL-S, WSMO [155] and Features & Properties in WSDL 2.0 and [4]. However, the properties they define are fixed and these approaches do not take into account business and industry-centric vocabulary. In contrast, flavoring aspects support flexible vocabulary specification and domain-independent constraint specification. In essence, service flavoring addresses variability in service offerings arising out of the need to differentiate services in the marketplace – variability in service definition and provisioning.

6.5 Summary

In this chapter we presented a strategy to support differentiation of services in a service marketplace. We identified terms of offer of a service to be the *changing part* in a service description in the context of differentiation. Certain terms of offer that make service offerings *attractive* to consumers become differentiating or flavoring aspects. New and attractive terms of offer could be associated with a service through service policies without requiring a change in the underlying service realization mechanisms. Hence, by

adopting the service flavors strategy, a service provider can unintrusively differentiate his services by creating competitive as well as targeted offerings.

Chapter 7

Service Consumer APIs

Addressing Heterogeneities in Service Consumption

In the previous chapters, our focus was on modeling and provisioning of both fine-grained and coarse-grained services from the perspective of the service provider. In this chapter, we address heterogeneities in service consumption, from a service consumer perspective. It is important for web-business platform owners (service providers) to support service consumption from different platforms as well as different types of customer applications. Supporting service consumption in heterogeneous consumer environments is a challenge, which nevertheless has to be addressed to increase platform usage.

Presently, the most popular approach of the web-business platforms such as eBay®, Google®, Amazon® and others has been to offer readily usable service consumer APIs for different technology platforms such as Java, .NET, Ruby and PHP [156]. Such an approach of providing technology platform-specific APIs is primarily to engage a wider developer community. In addition, supports service consumption by heterogeneous client applications such as web applications, desktop applications (widgets, dashboards etc.) and other packaged applications. In our opinion, it is a huge challenge for web-business platform owners to create, maintain and evolve consumer APIs in different programming languages targeting different technology platforms. In this chapter, we provide a model-driven development approach for service consumer APIs. Broadly, we derive a service consumption API model from our existing models (domain, resource and our services model) using model-to-model transformations. We then transform the generated consumption API model to client-libraries in different programming languages using model-to-text transformations.

7.1 Need for Service Consumer APIs

Before we deal with the need for service consumer APIs, let us establish the standard model in which a service is consumed. The WS-Arch (web service architecture) explains the general process of consuming a web service in detail [94]. Typically, the process consists of 2-steps (fig 7.1):

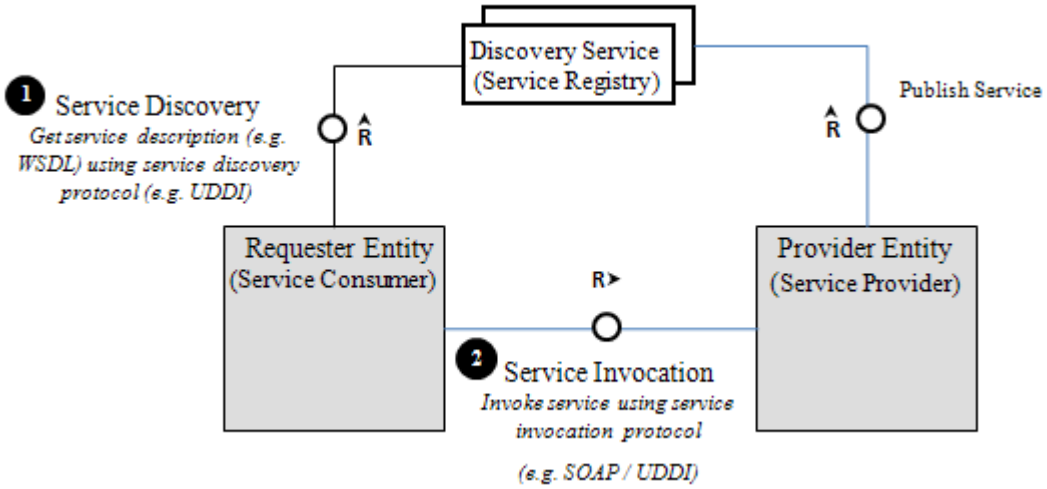


Fig 7.1: Process of Engaging a Service

Step 1 - Service Discovery: The requester entity (the service consumer) becomes aware of the existence of a service provided by a provider entity (the service provider). Awareness could be through service discovery supported by a service registry [157]. Technically, discovery is supported by a search and discovery protocol (e.g. UDDI) [158]. The service consumer then obtains a formal machine-readable service description (e.g. WSDL).

Step 2 – Service Invocation: The requester entity formulates a service invocation call using an invocation protocol (e.g. SOAP / HTTP) based on the formal service description. The interaction between the requester and provider entity is through loosely-coupled message exchanges.

Though webservices follow open-standards, service invocation for the consumer is not so straightforward due to heterogeneity in invocation protocols. For example, SOAP binding is just one invocation protocol for services defined using WSDL. There are other such bindings such as direct binding with HTTP, RMI/IIOP or Java Connector API [159]. In case of REST services, the invocation protocol could be plain HTTP (HTTPS in case of secure invocations). Currently service invocation approaches are of 2-types:

- *Direct Consumption:* Developers could assemble a SOAP message (or for that matter, any RSS/APP or Plain-XML messages) based on the service description and send it directly across the network to the service endpoint. However, direct consumption fails to hide low-level communication and data encoding details from programmers.

- *Consumption using automatically generated consumer proxies:* A proven approach is to automatically generate a client-proxy in the target programming language (using existing tools) and use the proxy to consume services from client programs. This approach is quite popular and is based on the well-known Proxy design pattern [160]. The advantage of using the automatically generated proxy is that consumers are not required to deal with XML messaging and other low-level communication details. They can consume services through the native programming paradigm. For example, in an object-oriented environment like Java, service interfaces can be viewed as classes, service operations as methods and faults as exceptions [161].

7.1.1 Advantages of Service Consumption APIs

Though automatically generated client-proxies support service consumption in the target technology platform, they still need to be wrapped around in client libraries (or service consumption APIs, as we call them). This is also the most prevalent approach among the existing web-business platforms. There are certain advantages of creating service consumption APIs:

1. A service consumption API addresses particular business functionality unlike a client-proxy which addresses a single service. An individual client-proxy can be generated for a fine-grained MANAGEORDERSERVICE service with a SOAP-style interface. However, we would need to provide the customer with a client library for the entire 'Order

Management' business functionality. The order management client library would support, creating, changing and cancelling orders. In addition, it would support tracking of order, choosing a logistics provider and shipping the order. Hence, it is essential to group individual services addressing particular business functionality and to create a service consumption API around that business functionality. For example, eBay® provides separate developer APIs for different business functionality such as buying, selling and market research (tracking pricing trends etc.) [11] [162].

2. Service consumption API provides the semantic underpinning for consuming these services. They are essential for maintaining the conceptual integrity of the client application.

3. Service consumption API insulates customer applications from changes to automatically generated client-proxies. Client proxies could change due to changes in service description, new service versions as well as changes to protocol bindings.

Due to the advantages it offers, a service consumption API is the best proposition to address heterogeneities in invocation protocols in the client applications. Most importantly they bridge the gaps between the service-oriented paradigm with that of the native application's paradigm (object-oriented) [163], ensuring conceptual integrity of the client applications.

7.1.2 Challenges in Creating Service Consumption APIs

The service consumption APIs are part of the consumption layer of our 4-layer SOA architecture (fig 7.2). The consumption APIs support consumption of both coarse-grained and fine-grained services. Creating service consumption APIs is no doubt advantageous; however managing and evolving these APIs is not as easy. There are certain challenges that have to be addressed; the most important being the large effort required to create, maintain and evolve consumption APIs consistently for different technology platforms.

Consider the following scenario – based on a new business case, specific platform functionality may have to be exposed or enhanced. This would result in either creation of a new service (either fine-grained or coarse-grained), or in a new version of an existing service. Such a change would translate to either creation of new consumption APIs or

enhancement of an existing consumption API. Service consumption API for each technology platform must be modified in order to support this change. The APIs must also maintain backward compatibility. In addition, it requires a lot of effort to create new consumption APIs to cater to other technology platforms, say the platform-owner would want to support SAP® ABAP-based environments.

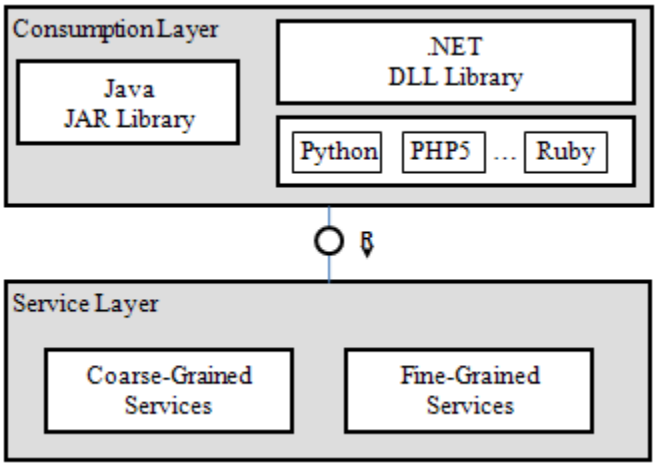


Fig 7.2: Service Consumption APIs in Consumption Layer

How do we enable the platform owner to handle these challenges? We propose a model-driven approach to support the creation and evolution of service consumption APIs. With this approach, we believe that the effort to create and evolve these APIs would significantly reduce. In addition, consumption APIs for new technology platforms can be developed easily either by the platform owner or by the community.

7.2 Our Model-Driven Development Approach

As mentioned earlier, we employ a model-driven development approach to develop and evolve service consumption APIs. We create a standard UML2 class diagram from our other MOF2-based models (domain, resources, and the services model). We leverage existing transformations to transform the UML2 class model to different languages such

as Java, .NET etc. Fig 7.3 shows the transformations that are involved in our approach. The model-to-model transformations from the domain, resources and the services models create a UML2 class model of the service consumption APIs. We use the standard UML2 class model for our service consumption APIs becasue we support an object-oriented client library.

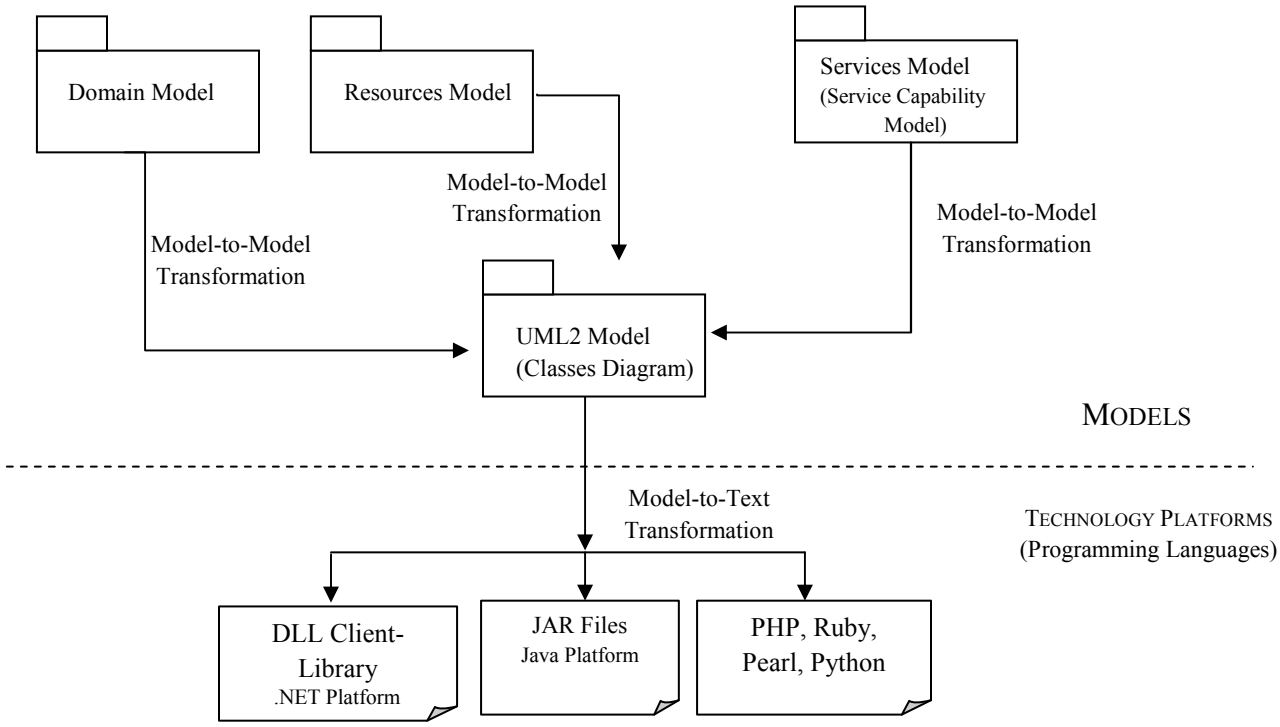


Fig 7.3: Model-Driven Approach to build Service Consumption APIs

Using the UML2 class model, we can create client libraries for different technology platforms. For creating client libraries in different programming languages, we rely on existing transformations such as ATL transformation for UML2-to-Java, UML-to-C# transformations for the .NET platform [164], and xmi2php for UML-to-PHP transformations [165].

For the model-to-model transformations, we could technically use any transformation language to transform our input models (domain, resources and the services model) to the UML2 model. In fig 7.4, we present our model-to-model transformations using a

UMLX diagram – a visual concrete syntax to MOF QVT[7] transformations – to explain our model-to-model transformations. Though our service consumption APIs are based on UML2 metamodel and our transformations are to UML2 Class Diagram model, we still introduce domain driven design notions in the consumption API model. We believe that the domain-driven design notions are important in order to provide a consistent mechanism to use consumption APIs for both the fine-grained and the coarse-grained services. We use a necessary set of five concepts namely *Domain Object*, *Repository*, *Factory* and a *Service* from the domain-driven design methodology in our consumption API model. We have a light-weight extension UML2 Profile for the Service Consumption API to represent these model concepts (fig 7.5).

Each *Entity* and *Aggregate* from the domain model is directly mapped to a *Domain Object* (basically a Class), if it is exposed as a resource, in the resource model. For simplicity, we do not distinguish between an *Entity* and an *Aggregate* in the consumption API model. We retain the name of the Resource (if there are any differences in name between Entity, Aggregate and the Resource). We also preserve the concept of a *Repository* and a *Factory* to support the life-cycle of the Domain Object in the consumption model. The repository supports basic CRUD operations through its methods *add, remove, update and findBy** on the domain objects. The *RepositoryItem* is also preserved in the consumption API model to handle repository operations. The major difference between the domain-model and the service consumption API model is the implementation.

[7] Though we depict our model-to-model transformations using a UMLX visual syntax, the diagram (fig. 7.3) is not completely compliant to UMLX concrete-syntax and is primarily used for representation purposes.

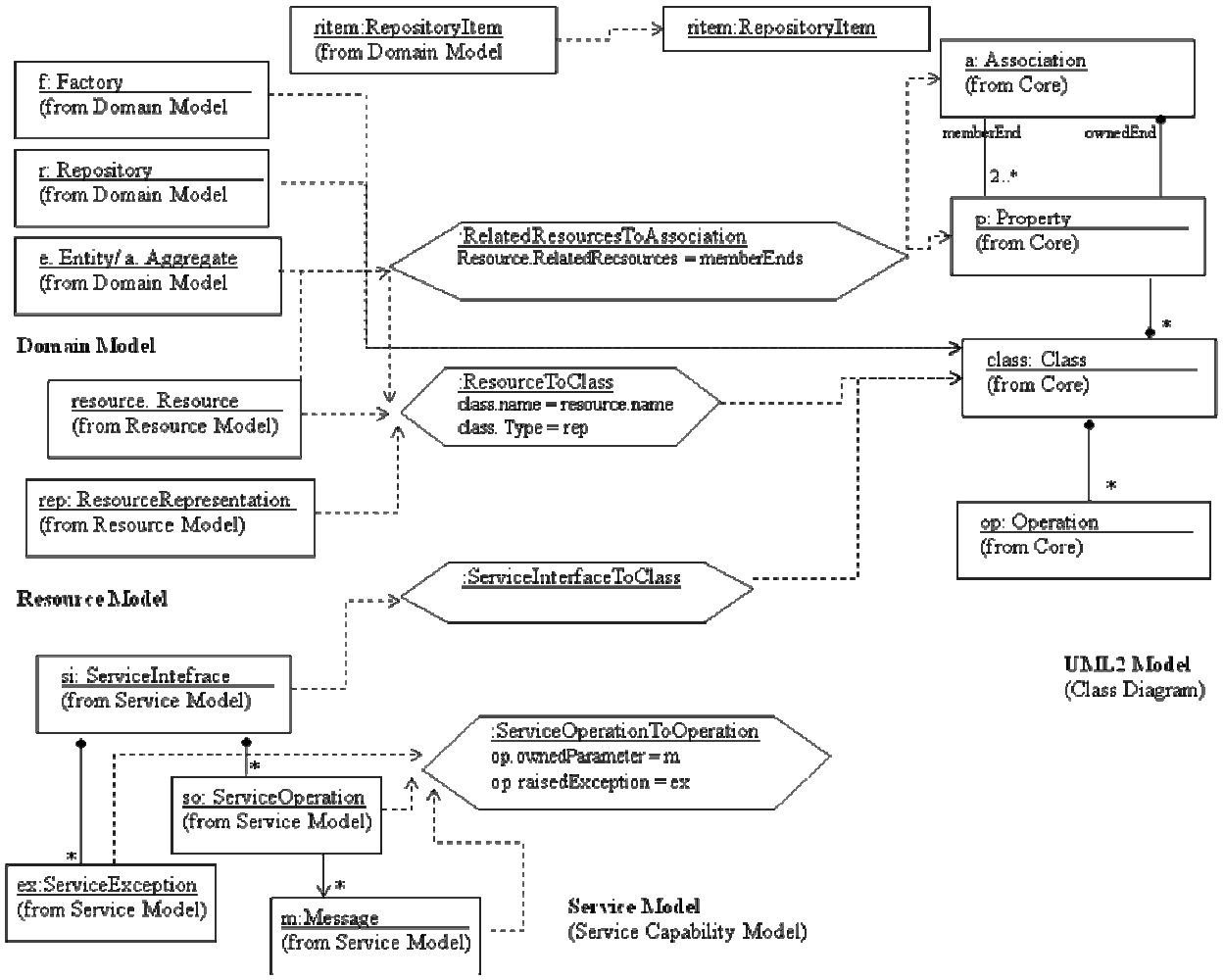


Fig 7.4: Model-to-Model Transformations to the UML2 Service Consumption API Model

This is in addition to certain concepts in the domain model which are missing in the service consumption API model. In the domain model, the focus of implementation code is provisioning; whereas, the focus of the consumption API model is to abstract client-proxy calls to the remote services. The repository methods consume the client-proxies of these CRUD services (either SOAP-based or REST-based interfaces), abstracting those details from the consumer. Thus, we support fine-grained manipulation of our resources through our service consumption API.

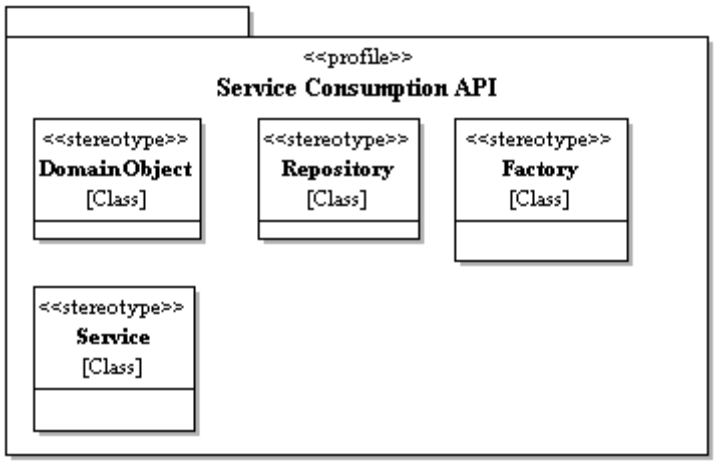


Fig 7.5: Light-weight UML2 Profile for Service Consumption API

For the coarse-grained services, the Service Interface in the Service Capability Model is transformed to a Class (*Service*), with the Service Operations and Message(s) as operations and parameters. The Service Exception becomes the *raisedException* of the class operation. In the next subsection, we would look at a few examples of service consumption API models.

7.2.1 Examples

Consider the online shopping example we presented in section 3.4, where the Purchase Order and the Product (OrderItem) was exposed as web resources. As mentioned earlier, this supports the customers to place orders directly from their systems instead of using the online shopping cart. By using our model-driven approach, it is possible to generate a service consumption API model for the *Direct Order* functionality (fig. 7.6). Our model-to-model transformations create a UML2-based consumption API model with *Domain Objects* – Purchase Order, OrderItem, a *Repository* and *Factory* for these domain objects. Using this UML model it is possible to create technology platform-specific client-libraries to aid service consumption through existing model-to-text transformations.

For an example of a coarse-grained service, consider the example of the AUCTIONITEM service (fig. 7.7). The AuctionItemInterface is a *Service* – transformed from the services

model. It has a single operation AuctionSingleItem (OP). The AuctionSingleItem (OP) operation has an AuctionRequestMessage (M) input parameters and would return an AuctionResponseMessage parameter. It also has AuctionNotPermitted (Ex) as a *raisedException* for the method AuctionSingleItem. The method implementation would call the automatically generated client-proxy in that particular programming language.

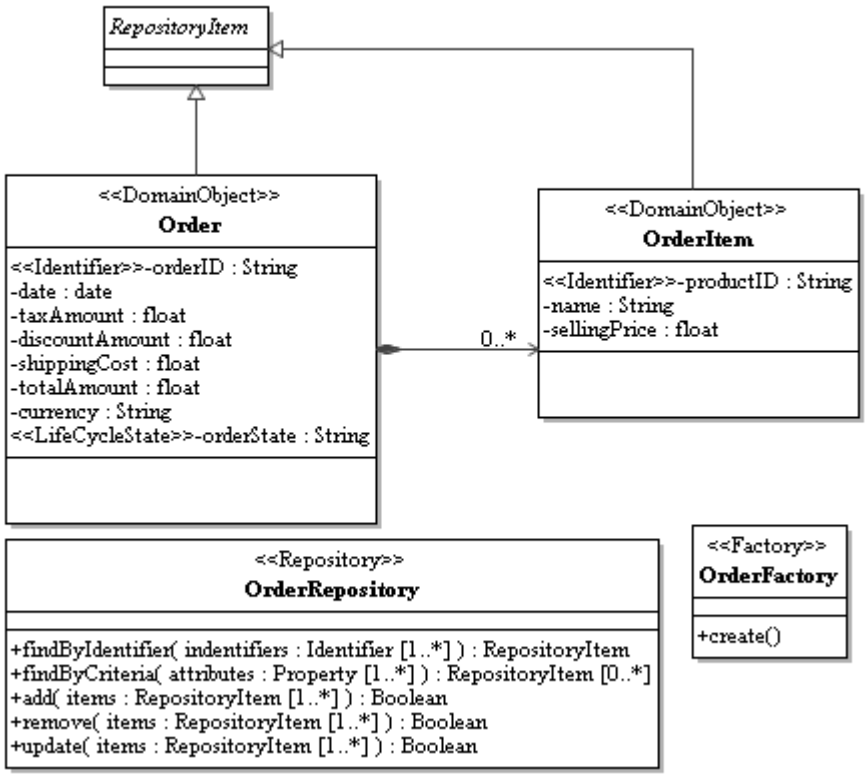


Fig 7.6: Partial Service Consumption API Model – Fine-Grained Service Access

Fig 7.7: Service Consumption API Model – Coarse-Grained Service

7.3 Related Work

The Web Service Invocation Framework (WSIF) [159] is a closely related work. WSIF is a framework based on Java to invoke webservices generically, independent of the invocation protocol (protocol bindings). WSIF uses an abstract WSDL description of a web service and allows developers to program against this abstract WSDL to access the remote service, independent of protocol bindings. New protocol bindings can be supported by the WSIF dynamic providers. Using the abstract WSDL, WSIF framework generates a stub – a Java object, which can be readily consumed in Java programs. Though WSIF solves the problem of invocation protocol heterogeneities in the consumption platform, it is specific for the Java platform.

Another interesting related work is REST Describe & Compile tool [166]. The goal of the tool is to generate client-code in various programming languages to access REST-based webservices, described using WADL. The REST Compile tool is a compiler which generates an Abstract Syntax Tree (AST) based on an input WADL file. Using the AST, the compiler generates client-code for consuming REST-based services in various programming languages (currently supports PHP 5, Ruby, Python and Java). The tool reduces considerable effort in creating and maintaining client-code of REST-based services access. Though REST Compile supports automatic client-code generation for REST-based fine-grained services described using WADL, it is quite similar to the numerous consumer proxy generation tools for WSDL in different technology platforms. In contrast, our service consumption API provides client-libraries compliant to the domain-driven design methodology. This supports conceptual integrity of consumer applications based on an object-oriented paradigm.

7.4 Summary

In this chapter, we support the creation of service consumption APIs to consume fine-grained and coarse-grained services. Using our service consumption APIs we address 2-important points in service consumption:

- Firstly, we address *heterogeneities in service consumption environments* arising due to different invocation protocols. We do this by providing a client-library to

abstract the application programmer from low-level communication and protocol details.

– Secondly, we support *Reduction in effort* involved in creating and evolving technology platform-specific APIs. We do this by adopting model-driven development principles to generate client-libraries in different programming languages.

Using our approach web-business platform owners can create client-libraries to consume services in different programming languages targeting various technology platforms.

Chapter 8

Epilogue

Evaluation, Experiences and Pragmatics

Our thesis addressed the research problems in the area of "opening-up" web-business platforms using webservices. Through this thesis, we have attempted to answer our succinct research questions. We support platform owners in methodically opening-up their web-business platforms through fine and coarse granular webservices using our *model-driven approach*. We represent service artifacts and metadata at a conceptual level using our *model views and metamodels*; and additionally, support competitively and unintrusively differentiating services in a services marketplace using our *service flavors strategy*.

8.1 Addressing Problem Areas

We had mentioned four problem areas in section 1.1. Our thesis addresses each of these problem areas. We present a correlation of how our thesis addresses those problem areas. Using our model-driven approach, we capture the solution space using high-level conceptual models; therefore, offering insulation from changing standards, supporting heterogeneities in the environment and increasing longevity of the solution. By doing this, we addressed the *Evolving Standards Problem*. For example, a service property can either be represented using Features & Properties in WSDL 2.0, or using separate service policies. In addition, a service can be offered as either a REST-based or a SOAP-based service as we demonstrated with the Shopping.com Listings example in fig. 3.2.

Our models – the services and resources model - capture metadata with a necessary representation depth irrespective of whether current standards support it. By doing so, we address the *Lean Service Metadata Problem*. For example, it is possible to capture service access information (registration information such as *developer key* and *merchant ID*) using formal service policies, which could be machine-processed leading to automatic consumption of services. A service could be defined as an *abstract service*, to represent intention to outsource, as we demonstrated with the VALIDATEAUCTIONITEM service in section 4.2. However, current standards do not support defining an abstract service.

Through our service flavoring approach – our method to create targeted and differentiated service offerings – we addressed the *Unintrusive Differentiation of Service Offerings Problem*. Service offerings can be competitively differentiated by unintrusively changing differentiating aspect as demonstrated in the *Subscription service flavor* and the *USFSB service flavor* examples in section 6.3.

We also created UML2 profiles of our model views in order to leverage existing modeling tools and skills. Our profiles provide visual modeling syntax for domain and business experts to define services, altogether removing the need to understand verbose XML syntax. Through our model views and corresponding profiles, we addressed the *Lack of Visual Syntax Problem*.

8.2 Criteria for the Solution

In chapter 1, we presented seven-point criteria. Any solution to the research problems we mentioned would satisfy these criteria. In this section, we provide a mapping between each criterion and evaluate how we address it in our thesis.

Criterion #1: The services must be represented at a conceptual and technology-agnostic level. *In order to insulate the service-oriented solution from technology changes, the solution must be captured at an abstract and conceptual-level agnostic to technology considerations during early-stage development. The service representation must describe both the capability-on-offer – the underlying business functionality – and the terms of offer of the service.*

Using our service and resource models we represent both coarse and fine-grained services at a conceptual and technology-agnostic level. While our service capability model presented in section 4.3.2 represents the capability on offer, our service policy models in section 5.2 represents the terms of offer of the service.

Criterion #2: The high-level conceptual service representation must be easily convertible to executable service specifications. *It must be possible to easily convert high-level conceptual service representations to executable service specifications, based on technical considerations like protocol and channel of access.*

The service representations captured in our models can be converted to executable specifications using model-to-text transformations. Such a transformation is possible because our metamodels are based on OMG's MOF2. We demonstrated in section 4.5, how our service capability model of the AUCTIONSERVICE was transformed to executable WSDL 2.0 specification using MTL mappings presented in Appendix I. In addition, the service provisioning code was generated for the Auction Manager in Java. In section 5.5, we demonstrated how service policy model was transformed to executable WS-Policy specifications. In section 3.3, we demonstrated how our resources model is used to generate service provisioning code as well as fine-grained service description as REST interfaces or using WSDL 2.0.

Criterion #3: The service representation method must have minimal concepts supporting maximal expressiveness. *By having minimal concepts with maximal expressiveness, business experts would find it easy to use the service representation method to describe various facets of services.*

Our services model is very well organized into six-model views representing different facets of services development. We have ensured that the model elements are minimal by creating our own domain-specific language (Services Metamodel, Resources Metamodel derived from domain-driven design metamodel) for modeling services, rather than using UML2, a general-purpose modeling language. Our approach of using domain-driven design techniques to provide a conceptual underpinning to our SOA-based solutions lowers the representation gap for domain experts. Expressivity of our models is proved by the successful mapping of models to executable specifications as demonstrated in sections 3.3, 4.5 and 5.5.

Criterion #4: The service representation should be used by different roles involved during early-stage services development. *The service representation must provide different views or perspectives for different roles to describe service artifacts during early-stage services development.*

In our thesis, we provide different model views for different stakeholders during early-stage services development. Our domain model, from which our resources model is built, is defined by domain experts using our domain model view. Similarly, we provide different model views such as service description, capability, policy, realization, mediation and deployment views. While the description view could be used by the business expert, the architect could use the capability view to describe service capability. The developer could use the realization view to define service provisioning. By providing different model views, we support different roles involved in services development.

Criterion #5: The service representation must have strong underpinnings in the application domain. *The service representation must have underpinnings in the application domain in order to support easy evolution of the solution and provide a common communication lingo between domain experts and the IT experts.*

Using the application domain as the conceptual underpinning, as advocated by the domain-driven approach, we aim to support solution evolution and provide a "ubiquitous language" for bridging the business as well as the IT experts. In addition, we hope to make services semantically rich by directly borrowing domain concepts.

Criterion #6: The service representation must be open-standards compliant and must leverage existing skill-sets and tools. *Our service representation method has to be based on open-standards and must leverage existing skill-sets in projects and popular tooling environments.*

Our service representation is based on models, as prescribed by the model-driven development approach. Our metamodels are based on MOF2; moreover we have provided corresponding UML2 profiles for our metamodel to leverage existing tooling. We also have Ecore (Eclipse EMOF Core) [167] representations of our metamodels (see section 8.5, Appendix II), which facilitate the use of model-driven environments such as eclipse-based openArchitectureWare [168] (a.k.a. oAW). Using such tools, it is easy to create transformations of our models to open-standards based executable specifications.

Criterion #7: The solution must support unintrusive changes to the commissioned services to support competitive differentiation. *The solution must support unintrusive changes to the already deployed (commissioned) services in order to differentiate service offerings from that of the competition in the services marketplace.*

Based on our service flavors strategy (chapter 6), we can create competitive services by creating either targeted or differentiated offerings. In order to do this, we support manipulation of *differentiating or flavoring aspects* of a service. These flavoring aspects are terms of offer, which improve the attractiveness of the service offering. We support service flavoring by attaching different service policies to existing services. As our policy enforcement points are abstracted from the service provisioning in the SOA-middleware, we can support unintrusive service differentiation.

8.3 Conformance

We evaluated our thesis on conformance to contemporary reference models and conceptual architectures. We wish to ensure compliance with the concepts presented in the OASIS Reference Model for Service-Oriented Architectures and the WS-Arch (Web services Architecture).

8.3.1 OASIS SOA Reference Model

We present the state of our compliance to the reference model's conformance guidelines (Section 4 of [32]) below:

1) Have entities that can be identified as services as defined by this Reference Model

We have a first-class model entity 'Service' in our services metamodel. Service represents a set of capabilities provided by a service provider (or a service aggregator) which meets the goals (needs) of service consumers.

2) Be able to identify how visibility is established between service providers and consumers

A service consumer could become aware of a service provider and its capabilities on offer through a service description (awareness). However we do not address discovery and advertising capabilities within a services marketplace as yet. Service providers and service consumers interact through an interaction point (reachability).

3) Be able to identify how interaction is mediated

The interaction is mediated through the understanding provided by the service description. The message exchange patterns of service operation dictate the sequence of communication between the provider and the consumer. In case, an interaction between a consumer and provider needs data or process mediation, our 'Mediator' supports it.

4) Be able to identify how the effect of using services is understood

Given that the pre-conditions and policies are met, the post-conditions on a service operation specify the real-world effects of invoking the service operation.

5) Have descriptions associated with services

Service description (containing the service interface, associated operations and properties) and the service policy provide description about choosing and using a particular service. While the service capability view describes the capability on offer, the service policy view represents the terms of offer.

6) Be able to identify the execution context required to support interaction

Though we have the infrastructural elements such as service description and service policies, we do not completely address all requirements of execution context to support interaction between the providers and consumers.

7) It will be possible to identify how policies are handled and how contracts may be modeled and enforced

Our service policy view completely addresses modeling of policy alternatives, assertions for a service. Enforcement of policy is possible through the policy enforcement point (PEP), a SOAP intermediary. Modeling support for service contracts is still missing.

8.3.2 WS-Arch (Web Services Architecture)

We also compare our models to W3C's Webservices Architecture [94]Webservices. In table 8.1, we present a comparison of the concepts present in the webservices architecture with the concepts in our metamodels. The webservices architecture represents the concepts and their relationships as concept maps; whereas, we have formal services and resources models based on MOF2. As our focus is on early-stage design, certain concepts (e.g. message body and header) are not present in our model.

Webservices Architecture	**Services/ Resources Metamodel**
Service Oriented Model	Service Definition / Service Capability View
Service	Service
Service Description	Service Description
Service Interface	Service Interface
Person / Organization	Ownership Domain
Provider Agent	Service Provider
Requestor Agent	Service Requestor
Service Intermediary	Service Mediator
Policy Model	Service Policy View
Policy	Service Policy
Policy Description	Service Policy
Domain	Domain Assertion
Permission / Obligation Guard	Policy Assertion
Resource Oriented Model	Resources View

Service (is a Resource)	Resource (fine-grained services)
Message Oriented Model	Service Capability View
Message	Message
Message Exchange Pattern	Service Operation's Message Exchange Pattern

Table 8.1 Related Concepts in the Webservices Architecture

8.4 Experiences

8.4.1 Experiences in using the Models

We used our models to extensively model popular, non-trivial e-Commerce scenarios. We borrowed our fictitious scenarios from hugely popular web-business platforms such as eBay®, Shopping.com, FedEx® and Google®. We modeled an online shopping scenario similar to the one supported by eBay ProStores®, using our domain-driven design metamodel in section 3.4. We modeled the online shopping domain and created a resources model from the domain model. We modeled various facets of eBay Auctions® scenario in chapter 4 using our services model, model views and profiles. Further, we modeled a Shipping scenario closely relating to FedEx® shipping scenario.

In our experience, our models were ***formal and expressive*** enough to capture a high-level ***conceptual*** view of the domain, domain objects, various facets of services and policies. With the UML2 profiles for these model views, we were able to ***leverage existing tools*** supporting and ***use existing modeling skills***. Especially, we used a community edition of MagicDraw 15.0 to model our scenarios (details in 8.5.2). Most importantly, the model views supported ***incremental and iterative development*** of services. The model views provided implicit logical steps to define and develop services. Starting with the domain model, helped to leverage domain semantics, evolve resources and develop straight-forward fine-grained CRUD services. In the next step, the services model views, the service description, capability, realization and the deployment views

provided us with a logical sequence to design, realize and deploy services. In our experience of using these models, they are ***comprehensible*** and ***highly coherent***.

In order to generate runtime artifacts such as code and executable specifications from our models, there was some effort involved in setting up the model-driven development platform. However, this was a one-time evaluation and setup effort. We used existing tools to convert our MOF2-compliant models to EMF metamodel (see section 8.5.2). Writing transformations involved deeper understanding of current model-driven development practices, metamodels and transformation languages. While we wrote transformations to popular WSDL 2.0 and WS-Policy standards (see Appendix I), we realized that it was easy to map our models to executable specifications. In our opinion, the ***executability*** – the ease at which models can be executed to create runtime artifacts is good.

8.4.2 Experiences with Existing Tools

In order to work with these models, we needed a model-driven development platform. Such a platform should support creating model instances, validating constraints and finally support code-generation. We built our formal MOF2-based metamodels – our domain-driven design metamodel, resources metamodel, services metamodel and our consumption API metamodel – using a formal modeling tool, MagicDraw® [169]. Using the export feature, we exported our metamodels into XMI format (XML Metadata Interchange) (see Appendix II) (fig 8.1).

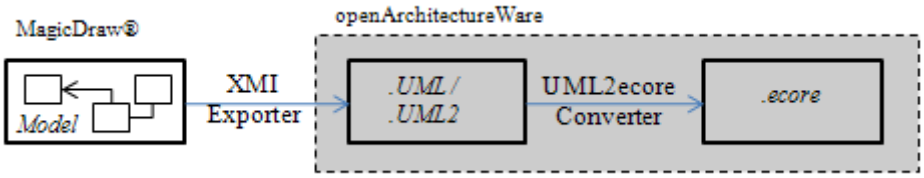


Fig 8.1 Transforming Model to .ecore format

For manipulating our models, we used openArchitectureWare (oAW), a model-driven development platform. oAW is part of the Eclipse GMT (Generative Modeling Technologies) [170] project to support instantiation and creation of model, transformation to other models and code. oAW is available as a plug-in in the open-source eclipse development environment. oAW, as well as other tools are converging on the ECore or the EMF metamodel standard to represent metamodels.

The EMF metamodel is defined by the EMF (Eclipse Modeling Framework) project and is based on Essential MOF, a core subset of the MOF2 standard. Using a model-to-model transformation (UML2Ecore), we transformed our metamodel (in XMI) to *.ecore* format EMF metamodels. For example, we transformed our domain-driven design metamodel to DomainDrivenDesign.ecore (fig 8.2). We present other *.ecore* models in Appendix II. Using the *.ecore* format of the model, we could manipulate the model, create validated model instances and write transformations to generate code.

In our experience, it was fairly simple and easy to convert our MOF2-based metamodels to EMF metamodels, manipulate them and transform them to executable specifications and code. The advantage of using a mature model-driven development platform is that there are community developed cartridges (transformations) which we could readily leverage, for example, the JavaBasic cartridge supports service provisioning code generation in Java. There are other comparable MDA tools such as AndroMDA [171], Motion Modeling [172] etc. As most, currently not all, of the MDA tools support EMF metamodel we could safely assume that our models could be used with those tools as well.

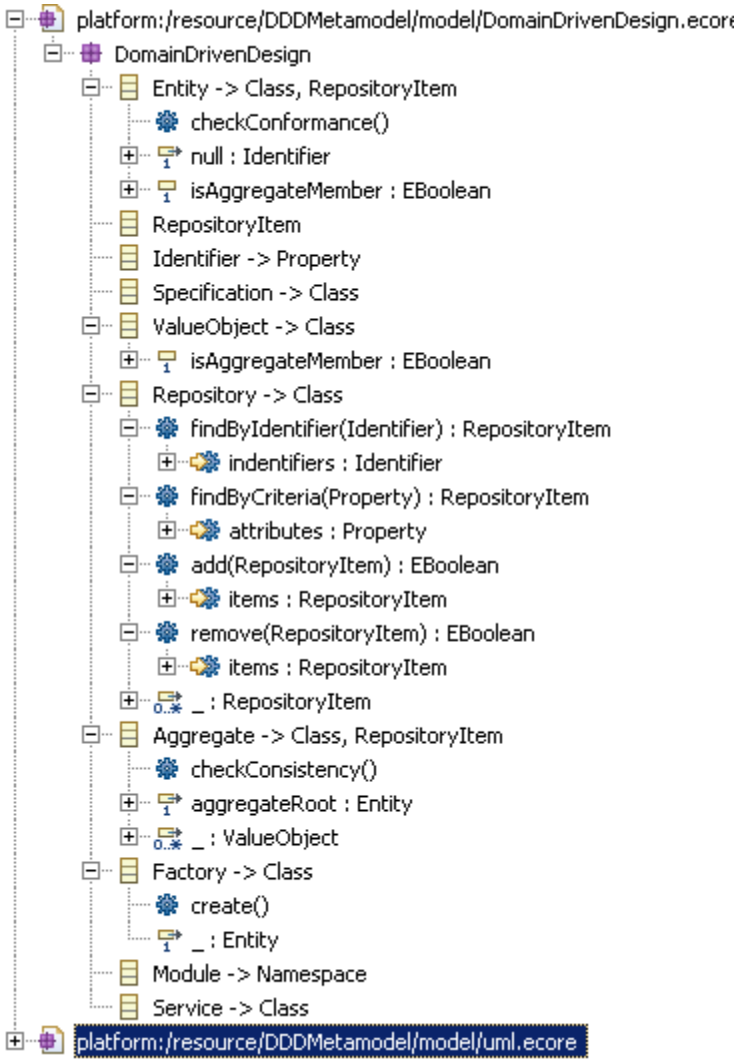


Fig 8.2 Domain-Driven Design .ecore format

8.5 Pragmatics & Future Work

In this section we discuss certain pragmatic issues, we realized while using our approach. We mention some limitations and possible solutions leading to future work. Below is a set of pragmatic issues:

***Model Extensions*:** Though we have tried our best to create metamodels which capture the domain pretty well, we do envisage scenarios where we would need to add new attributes or model elements to our existing metamodels. In particular, we envision two scenarios *a) Development-specific model attributes* – to support the service development process and *b) Domain-specific Model elements or attributes*. An example for development-specific model attributes would be addition of a *"versionID"* field to our "Service" model element to support version tracking or addition of attributes to model elements which would help software asset tracking in a development environment. For development specific model attributes, we propose to have a *DevelopmentDescriptor* model element, associated with each model element at the metamodel which would have the necessary attributes. For handling domain-specific model elements or attributes we would have to evaluate *model versioning techniques* [173].

Support for Semantics: Currently our models do not address semantics explicitly, though there is lot of traction on semantic webservices in research. Current approaches for adding semantics is converging on annotating service descriptions with semantics, especially the Semantic Annotations for webservices (SAWSDL) [174] based on WSDL-S [175]. In the future, we could look at using MOF2-compliant ODM (Ontology Definition Metamodel) in our platform to work with ontologies. We would evaluate *model annotation* techniques [176] to annotate our models with these ontology-defined semantics. In the policy front, we could evaluate the use of SBVR (Semantics of Business Vocabulary and Rules) [177] to define policy domain and vocabulary. SBVR also has a textual-syntax which is easy for business experts to work with.

Middleware-Agnostic Service Policy Enforcement: We currently achieve abstraction and modularity in policy enforcement using the PEP intermediary. This approach is tightly coupled to the underlying SOA middleware. However we would like to investigate a middleware-agnostic approach based on aspect-oriented techniques to support quantification and enforcement of service policies [136, 140]. In addition, we would have to investigate the issue of modeling dependencies between policy domains, e.g. modeling the relation between service pricing and promotions.

8.6 Conclusions

This thesis is an attempt to address the problems faced by web-business platform owners with a service-oriented platform strategy. We strongly believe that our research would support platform-owners in systematically "opening-up" their web-business platforms and support large-scale platform adoption in the community. We believe our primary contribution is our model-driven approach. This includes our:

- *model views* which support systematic development of fine as well as coarse-grained services,
- standards-compliant *metamodels* which support the actual modeling of service artifacts (presented in chapter 3, 4, and 5)
- *Service flavoring strategy* (presented in chapter 6) to differentiate service offerings in a services marketplace.

We demonstrated with non-trivial examples from popular online businesses, that our approach is suitable to realize the platform strategy of any web-business platform. As next steps, we would look at commercialization opportunities of the approaches mentioned in this thesis. Specifically, we would look to build an eclipse-based model-driven development platform to support platform-owners in using services to open-up their web-business platforms.

References

[1] L.H. Lin, A. Tanyavutti, and S. Jindrapacha, "Analyzing eBay Platform Strategies: An Application of Meyer's Product Platform Strategy Model," *Management of Engineering and Technology, Portland International Center for*, 2007, pp. 125-142.

[2] D.S. Evans, A. Hagiu, and R. Schmalensee, *Invisible Engines: How Software Platforms Drive Innovation and Transform Industries*: The MIT Press, 2008.

[3] A. Gawer and M.A. Cusumano, *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*: Harvard Business School Press, 2002.

[4] M.E. McGrath, *Product Strategy for High-Technology Companies: How to Achieve Growth, Competitive Advantage, and Increased Profits*: Irwin Professional Publishing, 1994.

[5] M.H. Meyer and A.P. Lehnerd, *The Power of Product Platforms*: Free Press, 1997.

[6] H. Jegadeesan and S. Balasubramaniam, "A MOF2-based Services Metamodel", in Journal of Object Technology, vol. 7, no. 8, Nov-Dec 2008 (to appear).

[7] J.J. Moreau et al., *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, 2006: http://www.w3.org/TR/wsdl20/ [September 20, 2008].

[8] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, SOAP Version 1.2, World Wide Web Consortium(W3C), Working Draft WD-soap12-20010709, July 2001

[9] M.P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Communications of the ACM*, vol. 46, 2003, pp. 25-28.

[10] R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures". Doctoral Thesis, University of California, Irvine , 2000.

[11] "eBay Developers Program". Internet: http://developer.ebay.com/, [September 20, 2008]

[12] H. Jegadeesan and S. Balasubramaniam, "Differentiating Commoditized Services in a Services Marketplace," *IEEE International Conference on Services Computing (SCC'08.), 2008*, pp. 153-160

[13] D.C. Fallside, "XML Schema Part 0: Primer," *W3C Candidate Recommendation CR-xmlschema-0-20001024, World Wide Web Consortium (W3C), Oct*, 2000.

[14] M.P. Papazoglou et al., "Service-Oriented Computing: A Research Roadmap," International Journal of Cooperative Information Systems (IJCIS), vol.17, Issue 2 (June 2008), pp. 223-255, 2008.

[15] M.P. Papazoglou et al., "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, pp. 38-45, 2007,

[16] D.S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*: Wiley, 2003.

[17] A.G. Kleppe, J.B. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*: Addison-Wesley, 2003.

[18] M.P. Gervais, "Towards an MDA-oriented methodology," *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 265-270.

[19] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An Architectural Framework for Service Definition and Realization," *Proceedings of the IEEE International Conference on Services Computing table of contents*, IEEE Computer Society Washington, DC, USA, 2006, pp. 151-158.

[20] O.M.G. (OMG), *Meta Object Facility (MOF) Specification 2.0 Core*, 2006. Internet: http://www.omg.org/spec/MOF/2.0/. [September 20, 2008]

[21] T. Gardner et al., "A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard," *MetaModelling for MDA Workshop*, 2003.

[22] J. Oldevik et al., "Toward Standardised Model to Text Transformations," *Lecture Notes In Computer Science*, vol. 3748, 2005, p. 239.

[23] T. Oreilly, "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software."

[24] T. Berners-Lee and M. Fischetti, *Weaving the Web*: Orion Business Books, 1999.

[25] M. Lawson, "Berners-Lee on the read/write web," *BBC NEWS*. Internet: http://news.bbc.co.uk/1/hi/technology/4132752.stm, 2005. [September 20, 2008]

[26] M. Muffatto, "Introducing a platform strategy in product development," *International Journal of Production Economics*, vol. 60, 1999, pp. 145-153.

[27] T.W. Simpson et al., "Platform-Based Design and Development: Current Trends and Needs in Industry," *ASME Design Engineering Technical Conferences-Design Automation Conference, Philadelphia, PA*, 2006, pp. 10-13.

[28] D. Butler, "Mashups mix data into global service," *Nature*, vol. 439, 2006, pp. 6-7.

[29] D.S. Platt, *Introducing the Microsoft .NET Platform*: Microsoft Press, 2001.

[30] D. Kramer, "The Java Platform: A White Paper," *Sun Microsystems Inc*, 1996.

[31] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling The Web Services Web: An Introduction to SOAP, WSDL, And UDDI," Internet Computing, IEEE, vol. 6, no. 2, pp. 86-93, 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=991449

[32] (OASIS), *Reference Model TC: Oasis Reference Model for Service Oriented Architectures (working draft 10)*, Technical Report, Organization for the Advancement of Structured Information Standards (OAIS), 2005.

[33] H. Jegadeesan and S. Balasubramaniam, "A Model-Driven Approach to Service Policies", in Journal of Object Technology, vol. 8, no. 3, Mar-Apr 2009 (to appear)

[34] E. Newcomer and G. Lomow, *Understanding SOA with Web Services (Independent Technology Guides)*: Addison-Wesley Professional, 2004.

[35] M. Henkel and J. Zdravkovic, "Approaches to Service Interface Design," *Proceedings of the Web Service Interoperability Workshop, First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005), Hermes Science Publisher, Geneva, Switzerland*, 2005.

[36] D. Box et al., "SOAP: Simple Object Access Protocol," *MSDN Library*, 2001.

[37] *"JavaScript Object Notation (JSON)". Internet: http://www.json.org.[September 20, 2008]*

[38] H. Wittenbrink, *RSS and Atom: Understanding And Implementing Content Feeds And Syndication*: Packt Publishing, 2005.

[39] "Zap Think Research - Solving the Service Granularity Challenge". Internet: http://www.zapthink.com/report.html?id=ZAPFLASH-200639. [September 20, 2008]

[40] N. To, "Introducing Service-Oriented Architecture": *Pro WCF*, Apress, 2007.

[41] F. Leymann, "Web Services: Distributed Applications without Limits": *Business, Technology and Web, Leipzig*, 2003.

[42] J.W. Yoder, R.E. Johnson, and Q.D. Wilson, "Connecting Business Objects to Relational Databases," *Urbana*, vol. 51, p. 61801.

[43] D. Szepielak, "REST-based Service Oriented Architecture for Dynamically Integrated Information Systems," *PhD Symposium at ICSOC*, 2006.

[44] D. Sprott and L. Wilkes, "Understanding Service-Oriented Architecture," *CBDI Journal*, 2004.

[45] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*: Prentice Hall, 2004.

[46] B. Benatallah et al., "Developing Adapters for Web Services Integration," *Lecture Notes in Computer Science*, Springer, 2005, pp. 415-429; Available : http://www.springerlink.com/content/qcgdyp6ja7xcwnck/.

[47] H.R.M. Nezhad et al., "Web Services Interoperability Specifications," IEEE *Computer*, 2006, pp. 24-32.

[48] S. Kumar and K. Padmanabhuni, "WS-I Basic Profile: a practitioner's view," *Web Services, 2004. Proceedings. IEEE International Conference on*, 2004, pp. 17-24.

[49] Steve Vinoski, "REST Eye for the SOA Guy," , *IEEE Internet Computing*, vol. 11, 2007, pp. 82-84.

[50] Pautasso, C., Zimmermann, O., and Leymann, F. 2008. Restful web services vs. "big"' web services: making the right architectural decision. In Proceeding of the 17th international Conference on World Wide Web (Beijing, China, April 21 - 25, 2008). WWW '08. ACM, New York, NY, 805-814

[51] W. Iverson, *Real World Web Services*: O'Reilly Media, Inc., 2004.

[52] A. Arsanjani, "Service-Oriented Modeling and Architecture," *IBM developer works*: Internet: http://www.ibm.com/developerworks/library/ws-soa-design1/, 2004. [September 20, 2008]

[53] U. Zdun, C. Hentrich, and W.M.P. Van Der Aalst, "A survey of patterns for Service-Oriented Architectures," *International Journal of Internet Protocol Technology*, vol. 1, 2006, pp. 132-143.

[54] L. Liu, S. Thanheiser, and H. Schmeck, "A Reference Architecture for Self-organizing Service-Oriented Computing," *Lecture Notes In Computer Science*, vol. 4934, 2008, p. 205.

[55] J. Cheesman and G. Ntinolazos, "The SOA Reference Model," *CBDi Journal*, 2004.

[56] A. Arsanjani et al., "S3: A Service-Oriented Reference Architecture," *IT Professional*, 2007, pp. 10-17.

[57] C.I.O. Council, "Federal Enterprise Architecture Framework Version 1.1,", *Internet:*, http://www.whitehouse.gov/omb/egov/a-1-fea.html. [September 20, 2008]

[58] S. Kent, "Model Driven Engineering," *Lecture Notes In Computer Science*, 2002, pp. 286-298.

[59] (OMG), "MDA Guide," *Version 1.0.1*, vol. 1, 2003, pp. 2003-05. Available: http://www.omg.org/docs/omg/03-06-01.pdf

[60] W. Zhang et al., "Transformation from CIM to PIM: A Feature-Oriented Component-Based Approach," *Lecture Notes In Computer Science*, vol. 3713, 2005, p. 248.

[61] A. van Deursen and J. Visser, "Domain-specific languages: an annotated bibliography," *ACM SIGPLAN Notices*, vol. 35, 2000, pp. 26-36.

[62] (OMG), "UML 2.0 Infrastructure Specification," *OMG formal document*, pp. 03-09, 2007. Available: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/

[63] K. Duddy, "UML2 must enable a family of languages," *Communications of the ACM*, vol. 45, 2002, pp. 73-75.

[64] J.M. Alvarez, A. Evans, and P. Sammut, "Mapping between Levels in the Metamodel Architecture," *Lecture Notes In Computer Science*, 2001, pp. 34-46.

[65] J. Poole and D. Mellor, *Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration*: John Wiley & Sons, Inc. New York, USA, 2001.

[66] D. Djuric, D. Gasevic, and V. Devedzic, "A MDA-based Approach to the Ontology Definition Metamodel," *Proceedings of a 4TH Workshop On Computational Intelligence And Information Technologies. October*, 2003.

[67] S. Cook, "The UML Family: Profiles, Prefaces and Packages," *Lecture Notes In Computer Science*, 2000, pp. 255-264.

[68] A.G. Kleppe, J.B. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*: Addison-Wesley, 2003.

[69] T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, 2006, pp. 125-142.

[70] F. Jouault and I. Kurtev, "Transforming Models with ATL," *Lecture Notes In Computer Science*, vol. 3844, 2006, p. 128.

[71] M. Kay, "XSL Transformations (XSLT) Version 2.0," *W3C Working Draft*, vol. 16, 2002.

[72] T.J. Grose et al., *Mastering XMI: Java programming with XMI, XML, and UML*, Wiley, 2002.

[73] R. Cover, *XML Metadata Interchange (XMI)*, 2001. Available: http://www.omg.org/technology/documents/formal/xmi.htm

[74] F. Chauvel and F. Fleurey, *Kermeta Language Overview*. Available: http://www.kermeta.org/docs/KerMeta-MetaModel.pdf

[75] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003.

[76] A. Kalnins, J. Barzdins, and E. Celms, "Model Transformation Language MOLA," *Lecture Notes In Computer Science*, vol. 3599, 2005, p. 62.

[77] E.D. Willink, "UMLX: A graphical transformation language for MDA," *Model Driven Architecture: Foundations and Applications*, 2003, pp. 03-27.

[78] A. Agrawal, "Graph Rewriting And Transformation (GReAT): A Solution For The Model Integrated Computing (MIC) Bottleneck," *18th IEEE International Conference on Automated Software Engineering (ASE'03)*.

[79] M. Albert et al., "Model to Text Transformation in Practice: Generating Code from Rich Associations Specifications," *Lecture Notes In Computer Science*, vol. 4231, 2006, p. 63.

[80] R.T. Fielding and R.N. Taylor, "Principled design of the modern Web architecture," *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, pp. 407-416.

[81]D. Booth et al., "Web Services Architecture," *W3C Working Group Note*, vol. 11, 2004, pp. 2005-1.

[82] D. HinchCliffe, "REST vs. SOAP: The Battle of the Web Service Titans," *Internet: http://soa.sys-con.com/node/79282*. [September 20, 2008]

[83] T. O'Reilly, "REST vs. SOAP at Amazon", Internet: http://www.oreillynet.com/pub/wlg/3005 [September 20, 2008]

[84] "Google Base"; Internet: http://www.google.com/base. [September 20, 2008]

[85] "Google Product Search"; Internet: http://www.google.com/products. [September 20, 2008]

[86] "Google Data APIs - Google Code"; Internet: http://code.google.com/apis/gdata/. [September 20, 2008]

[87] R. Sayre, "Atom: The Standard in Syndication," *IEEE Internet Computing*, 2005, pp. 71-78.

[88] "Google Checkout"; Internet: http://checkout.google.com. [September 20, 2008]

[89] "Google AdWords"; Internet: http://adwords.google.com/. [September 20, 2008]

[90] "What is the eBay API? — eBay Developers Program"; Internet: http://developer.ebay.com/common/api/. [September 20, 2008]

[91] E. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2003.

[92] C. Larman, *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design*, Prentice Hall PTR, 1998.

[93] N.A.B. Gray, "Comparison of Web Services, Java-RMI, and CORBA service implementations," *School of Information Technology & Computer Science, University of Wollongong*.

[94] W. (W3C), *Web Services Architecture*, 2004; Available: http://www.w3.org/TR/ws-arch/.

[95] M.J. Hadley, "Web Application Description Language (WADL)," *Technical Specification, Sun Microsystems*, 2006.

[96] J. Amsden, "UML Profile and Metamodel for Services", *OMG formal document soa/06-09-09*.Available: http://www.omg.org/cgi-bin/doc?soa/2006-9-9

[97] J. Sztipanovits and G. Karsai, "Model-integrated computing," IEEE *Computer*, vol. 30, 1997, pp. 110-111.

[98] W. Iverson and T. O'Reilly, "Web Services in Action: Integrating with the eBay Marketplace," *A White Paper from O'Reilly Media*.

[99] R. Housley et al., "RFC3280: Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," *RFC Editor United States*, 2002.

[100] J.B. Murphy, "Introducing the North American Industry Classification System," *Monthly Labor Review*, vol. 121, 1998, p. 43.

[101] P. Yendluri et al., "Web Services Policy 1.5-Primer", W3C Working Group Note, November 2007. Available: http://www.w3.org/TR/ws-policy-primer/

[102] A.S. Vedamuthu et al., "Web Services Policy 1.5-Attachment," *W3C Recommendation*, September 2007. Available: http://www.w3.org/TR/ws-policy-attach/

[103] S. Efftinge and C. Kadura, *OpenArchitectureWare 4.1 Xpand Language Reference*; Internet:http://www.eclipse.org/gmt/oaw/doc/4.1/r20_xPandReference.pdf [September 20, 2008]

[104] C. Emig et al., "Model-Driven Development of SOA Services," Technical report, Forschungsbericht, 2007.

[105] M. Brambilla et al., "Model-driven Development of Web Services and Hypertext Applications," *SCI2003, Orlando, Florida, July*, 2003.

[106] K. Bai'na et al., "Model-Driven Web Service Development," *Advanced Information Systems Engineering: 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004: Proceedings*, 2004.

[107] J. Bezivin et al., "Applying MDA Approach for Web Service Platform," *EDOC, IEEE Computer Society*, 2004, pp. 58–70.

[108] G. Benguria et al., "A Platform Independent Model for Service Oriented Architectures," *Proceedings of I-ESA Conference*, 2006.

[109] D. Skogan, R. Gronmo, and I. Solheim, "Web Service Composition in UML," *8th IEEE Intl. Enterprise Distributed Object Computing Conference*, pp. 47–57.

[110] R.T. Sanders et al., "Using UML 2.0 Collaborations for Compositional Service Specification," *Lecture Notes In Computer Science*, vol. 3713, 2005, p. 460.

[111] J. Bezivin et al., "An Experiment in Mapping Web Services to Implementation Platforms," *Research Report. Atlas Group, INRIA and LINA University of Nantes*, 2004.

[112] B. Bordbar and A. Staikopoulos, "Automated Generation of Metamodels for Web Service Languages," *Proc. of Second European Workshop on Model Driven Architecture (MDA)*, 2004.

[113] D. Frankel and J. Parodi, "Using Model-Driven Architecture to Develop Web Services," *IONA Technologies white paper*, 2002.

[114] R. Amir and A. Zeid, "A UML profile for service oriented architectures," *Conference on Object Oriented Programming Systems Languages and Applications*, ACM New York, NY, USA, 2004, pp. 192-193.

[115] R. Heckel, M. Lohmann, and S. Thone, "Towards a UML Profile for Service-Oriented Architectures," *Proc. of Workshop on Model Driven Architecture: Foundations and Applications (MDAFA), CTIT Technical Report TR-CTIT-03-27. University of Twente, Enschede, The Netherlands*, 2003.

[116] S. Johnston, "UML 2.0 Profile for Software Services," *IBM developerWorks*, vol. 15, 2005. Available: http://www.ibm.com/developerworks/rational/library/05/419_soa/

[117] M.P. Gervais, "Towards an MDA-oriented methodology," *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 265-270.

[118] B. Hofreiter, C. Huemer, and K.D. Naujok, "UN/CEFACT's Business Collaboration Framework-Motivation and Basic Concepts," *Proc. of MKWF04 Track on Coordination in Value Creation networks/Agent Technology for Business Applications, LNI GITO*, 2004.

[119] B. Korherr et al., "UN/CEFACT'S Modeling Methodology (UMM): A UML Profile for B2B e-Commerce," *Lecture Notes In Computer Science*, vol. 4231, 2006, p. 19.

[120] M. Broy, I.H. Krüger, and M. Meisinger, "A formal model of services," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 16, 2007.

[121] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF," *Electronic Commerce Research and Applications*, vol. 1, 2002, pp. 113-137.

[122] H. Lausen, A. Polleres, and D. Roman, "Web Service Modeling Ontology (WSMO)," *W3C Member Submission*, vol. 3, 2005. Available: http://www.wsmo.org/

[123] J. de Bruijn et al., "The Web Service Modeling Language WSML," *WSML Final Draft D*, vol. 16, 2005.Available: http://www.wsmo.org/wsml/

[124] Ü. Yalçinalp et al., *Web Services Policy 1.5 - Framework*, W3C Recommendation, 2007. Available: http://www.w3.org/TR/ws-policy/

[125] A.H. Anderson, "Domain-Independent, Composable Web Services Policy Assertions," *Proc. of the 7th IEEE Int'l Workshop on Policies for Distributed Systems and Networks, IEEE CS*, 2006, pp. 149–152.

[126] P.Walmsly et al., "XML Schema Part 0: Primer Second Edition," *W3C Recommendation*, 2004. Available: http://www.w3.org/TR/xmlschema-0/

[127] M.K. Smith, C. Welty, and D.L. McGuinness, "OWL Web Ontology Language Guide," *W3C Recommendation*, vol. 10, 2004. Available: http://www.w3.org/TR/owl-features/

[128] H.R.M. Nezhad et al., "Web Services Interoperability Specifications," IEEE *Computer*, 2006, pp. 24-32.

[129] A. Anderson, *WS-PolicyConstraints: A domain-independent web services policy assertion language*, November, 2005.

[130] A. Anderson, "XACML-Based Web Services Policy Constraint Language (WS-PolicyConstraints)," *Working Draft*, vol. 5, 2005, p. 27.

[131] A.H. Anderson, "An introduction to the Web Services Policy Language (WSPL)," *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, 2004, pp. 189-192.

[132] S. Perera et al., "Axis2, Middleware for Next Generation Web Services," *Proceedings of ICWS 2006*, 2006, pp. 833-840.

[133] O.M.G. (OMG), *UML Profile and Metamodel for Services (UPMS) RFP*, 2006; http://www.omg.org/cgi-bin/apps/doc?soa/06-09-09.pdf.

[134] G. Ortiz, J. Hernandez, and P.J. Clemente, "How to Deal with Non-functional Properties in Web Service Development," *Web Engineering: 5th International Conference, ICWE 2005, Sydney, Australia, July 27-29, 2005: Proceedings*, 2005.

[135] G. Ortiz and J. Hernández, "Toward UML Profiles for Web Services and their Extra-Functional Properties," *Proc. Int. Conf. on Web Services, Chicago, EEUU, September*, 2006.

[136] G. Ortiz et al., "How to Model Aspect-Oriented Web Services," *Workshop on Model-driven Web Engineering*.

[137] J. O'Sullivan, D. Edmond, and A.H.M. ter Hofstede, *Formal description of non-functional service properties*, Technical report, Queensland University of Technology, Brisbane, 2005. Available from http://www. service-description. com, .

[138] G. Dobson, R. Lock, and I. Sommerville, "QoSOnt: a QoS Ontology for Service-Centric Systems," *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, 2005, pp. 80-87.

[139] F. Baligand and V. Monfort, "A concrete solution for web services adaptability using policies and aspects," *Proceedings of the 2nd international conference on Service oriented computing*, 2004, pp. 134-142.

[140] G. Ortiz and F. Leymann, "Combining WS-Policy and Aspect-Oriented Programming," *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, Washington, DC, USA: IEEE Computer Society, 2006, p. 143.

[141] J. Bosak, T. McGrath, and G.K. Holman, "Universal Business Language v2. 0," *Organization for the Advancement of Structured Information Standards (OASIS), Standard, December*, 2006.

[142] B.K. Gibb and S. Damodaran, *ebXML: Concepts and Application*, John Wiley & Sons, Inc. New York, NY, USA, 2002.

[143] Damodaran, S. 2004. B2B integration over the Internet with XML: RosettaNet successes and challenges. In Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers WWW Alt. '04. New York

[144] B. Medjahed et al., "Business-to-business interactions: issues and enabling technologies," *The VLDB Journal The International Journal on Very Large Data Bases*, vol. 12, 2003, pp. 59-85.

[145] R.H. Dolin et al., *HL7 Clinical Document Architecture, Release 2*, Am Med Inform Assoc, 2006.

[146] J.B. Stewart Jr, "Changing Technology and the Payment System," *Federal Reserve Bank of New York Current Issues in Economics and Finance*, vol. 6, 2000.

[147] P.R. Dickson and J.L. Ginter, "Market Segmentation, Product Differentiation, and Marketing Strategy," *Journal of Marketing*, vol. 51, 1987, pp. 1-10.

[148] J.L. Heskett, *Managing in the Service Economy*, Harvard Business SchoolPress, 1986.

[149] N.F. Noy and D.L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Knowledge Systems Laboratory, March*, 2001.

[150] V. Tosic, K. Patel, and B. Pagurek, "WSOL-Web Service Offerings Language," *Web Services, E-Business, and the Semantic Web: CAiSE 2002 International Workshop, WES 2002, Toronto, Canada, May 27-28, 2002: Revised Papers*, 2002.

[151] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, 2003, pp. 57-81.

[152] J. de Bruijn et al., "The Web Service Modeling Language WSML: An Overview," *Proceedings of the 3rd European Semantic Web Conference (ESWC)*, 2006.

[153] M. Tian et al., "A concept for QoS integration in Web services," *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on*, 2003, pp. 149-155.

[154] R.E. Filman, *Aspect-oriented Software Development*, Addison-Wesley, 2005.

[155] I. Toma and D. Foxvog, *Non-functional properties in Web services*; Available: http://www.wsmo.org/TR/d28/d28.4/v0.1/20061025/.

[156] D.M. Gosnell, *Professional Development with Web APIs: Google, eBay, PayPal, Amazon. com, MapPoint, FedEx*, Wrox Press Ltd. Birmingham, UK, UK, 2005.

[157] M.P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, Washington: IEEE Computer Society Press, December, 2003.

[158] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Addison-Wesley, 2002.

[159] M.J. Duftler et al., "Web Services Invocation Framework (WSIF)," *OOPSLA 2001 Workshop on Object-Oriented Web Services*, 2001.

[160] E. Gamma et al., *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Reading, MA, 1995.

[161] D.A. Chappell and T. Jewell, *Java Web Services*, O'Reilly Media, Inc., 2002.

[162] J.P. Mueller, *Mining eBay Web Services: Building Applications with the eBay API*, Sybex, 2004.

[163] N. Kothavenkata and S.R. Bhargava, "Object Orientation versus Service Orientation," *Workshop on Introducing Service-Oriented Computing WISOC 2007 Tempe, Spring 2007*.

[164] "ATL Transformations", Internet: http://www.eclipse.org/m2m/atl/atlTransformations/. [September 20, 2008]

[165] "XMI2PHP", Internet: http://xmi2php.sourceforge.net/. [September 20, 2008]

[166] "REST Describe & Compile"; Internet: http://tomayac.de/rest-describe/latest/RestDescribe.html. [September 20, 2008]

[167] F. Budinsky, S.A. Brodsky, and E. Merks, *Eclipse Modeling Framework*, Pearson Education, 2003.

[168] M. Volter, "openArchitectureWare 4–the flexible open source tool platform for model-driven software development," *Bericht, openArchitectureWare*, 2007.

[169] "MagicDraw - UML Modeling & Diagramming"; Internet: http://www.magicdraw.com/. [September 20, 2008]

[170] "Eclipse Generative Modeling Technologies (GMT) Project"; Internet: http://www.eclipse.org/gmt/. [September 20, 2008]

[171] "AndroMDA - MDA Tool", Internet: http://www.andromda.org/. [September 20, 2008]

[172] "Motion Modeling, open source MDA environment in eclipse"; Internet: http://motionmodeling.sourceforge.net/. [September 20, 2008]

[173] P. Hnetynka and F. Plášil, "Distributed versioning model for MOF," *Proceedings of the winter international synposium on Information and communication technologies*, Trinity College Dublin, 2004, pp. 1-6.

[174] J. Kopecký et al., "SAWSDL: Semantic Annotations for WSDL and XML Schema," *IEEE Internet Computing*, vol. 11, 2007, pp. 60-67.

[175] R. Akkiraju et al., "Web Service Semantics-WSDL-S," *W3C Member Submission*, vol. 7, 2005.

[176] S.J. Mellor et al., "Model-Driven Architecture," *Lecture Notes In Computer Science*, 2002, pp. 290-297.

[177] D. Chapin, "Semantics of Business Vocabulary & Business Rules (SBVR)," *Hawke et al.(2005)*, 2005.

Appendix I

Evidence for Empirical Evaluation: OMG Model-to-Text (MTL) Transformation from Services Model to WSDL 2.0

```
-------------------------------------------------------------------------------
//Transformation: Create WSDL 2.0 Description for a particular service endpoint
-------------------------------------------------------------------------------
@text-explicit
[template public ServiceModelToWSDL2Transformation(service : Service,
                                                   endpoint : InteractionPoint)]

[file ('file:\\'+service.name+'.wsdl', false)]

 <?xml version="1.0" encoding="utf-8" ?>
        <description>
            xmlns="http://www.w3.org/ns/wsdl"
            targetNamespace= [service.owner.namespaceURI/]
            xmlns:tns= [service.owner.namespaceURI/]
            xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
            xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

        <documentation>
                [service.realizedDescription.description/]
        </documentation>

        [messages: Set(Message = allMessages(service.realizedDefinition.interface)/]
        [CreateTypesDefinition(messages,service.owner)/]
        [CreateInterfaceAndOperationDefinition(service.realizedDefinition)/]
        [CreateBinding(service , endpoint)/]
        [CreateServiceEndpoint (service , endpoint)/]
        </description>


[/file]
[/template]
```

Listing 1: Transformation from Services Model to WSDL2.0

```
-------------------------------------------------------------------------------------------
//Transformation: Create Types definition
-------------------------------------------------------------------------------------------
@text-explicit
[template public CreateTypesDefinition(messages : Set(Message), od : OwnershipDomain)]
<types>
        <xs:schema
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace=[od.namespaceURI/]
                xmlns=[od.namespaceURI/]
        </xs:schema>
        [For (message:Message | messages)]
                [IterativeModelToXMLSchemaTransformation(message)/]
        [/For]

</types>
[/template]

-------------------------------------------------------------------------------------------
//Transformation: Iteratively transform Messages to XML Schema
-------------------------------------------------------------------------------------------
@text-explicit
[template public IterativeModelToXMLSchemaTransformation(messsage : Message)]
 ...
 ...
[/template]

-------------------------------------------------------------------------------------------
```

Listing 2: Create *Types* definition from messages by iteratively transforming Messages to XML Schema

```
-------------------------------------------------------------------------------------------
//Query: Get all Messages from supported Operations
-------------------------------------------------------------------------------------------
@text-explicit
[query public allMessages(si : ServiceInterface) :Set (Message)]
        [messages:Set(Message)/]
        [For (operation:Operation | si.supportedOperation]
                [For (message:Message | operation.message]
                        [messages: Set(Message) = Set{}| messages->union(message) /]
                [/For]
        [/For]
        [return messages/]
[\query]

-------------------------------------------------------------------------------------------
```

Listing 3: Query to get all messages from all supported operations

```
-------------------------------------------------------------------------------------------
//Transformation: Create Interface and Operation definitions
-------------------------------------------------------------------------------------------
@text-explicit
[template public CreateInterfaceAndOperationDefinition(description : ServiceDescription)]
        <interface name = [description.name/]>
                [CreateFaultDefinition(description.interface) /]
                [CreateOperationDefinition(description.interface) /]
        </interface>
[/template]

-------------------------------------------------------------------------------------------
//Transformation: Create Fault definition
-------------------------------------------------------------------------------------------
@text-explicit
[template public CreateFaultDefinition(interface : ServiceInterface)]
        [For (ex: ServiceException | interface.exception]
                <documentation>
                        [ex.description/]
                </documentation>
                <fault name = [ex.name/] element = [ex.type/]>
        [/For]
[/template]

-------------------------------------------------------------------------------------------
//Transformation: Create Operation Definition
-------------------------------------------------------------------------------------------
@text-explicit
[template public CreateOperationDefinition(interface : ServiceInterface)]
        [For (op: ServiceOperation |interface.supportedOperations)]
                <operation
                        name    = [op.name/]
                        pattern = [op.messageExchangePattern/]
                        style   = [op.style/]
                        wsdlx:safe = [op.isSafe/]>
                [let inputAssociationEnd = op.lookupAssociationEnd("input")/]
                [if not (inputAssociationEnd.oclIsUndefined())]
                        <input  messageLabel = "In" element = [op.input.type/]/>
                [/if]
                [let inFault = op.lookupAssociationEnd("infault")/]
                [if not (inFault.oclIsUndefined())]
                        <infault ref = [op.infault.qualifiedName/] messageLabel = "In">
                [/if]
                [let outputAssociationEnd = op.lookupAssociationEnd("output")]
                [if not (outputAssociationEnd.oclIsUndefined())]
                        <output messageLabel = "Out" element = [op.output.type/] />
                [/if]
                [if not (outFault.oclIsUndefined())]
                        <outfault ref = [op.outfault.qualifiedName/] messageLabel = "Out">
                [/if]
                </operation>
        [/For]
[/template]
```

Listing 4: Create interface and operation definitions

```
-------------------------------------------------------------------------------------------
//Transformation: Create Bindings
-------------------------------------------------------------------------------------------
@text-explicit
[template public CreateBinding(service : Service, endpoint : InteractionPoint)]
        [let interface : ServiceInterface = service.realizedDescription.interface/]
        <binding
                name = [endpoint.name/]
                type = [getBindingTypeURI(endpoint.bindingType)/]
                wsoap:protocol = [getBindingProtocol(endpoint.bindingType)/]>

        [For (fault :ServiceException | interface.exception)]
                <fault ref = [fault.qualifiedName/] wsoap:code = "soap:Sender"/>
        [/For]

        [For (op : ServiceOperation | interface.supportedOperations)]
                <operation ref = [op.qualifiedName/]
                           wsoap:mep = [getSOAPMEPURI(op.messageExchangePattern)/]>
        [/For]


        </binding>

        [function getBindingTypeURI(bindingType : BindingType) :URI]
                [{//return corresponding URI for binding type }/]
        [/function]
        [function getBindingProtocol(bindingType : BindingType) :URI]
                [{//return corresponding binding protocol for binding type }/]
        [/function]
        [function getSOAPMEP(mep :URI) :URI]
                [{//return corresponding SOAP message exchange pattern }/]
        [function]
[/template]
```

Listing 5: Create Bindings

```
-------------------------------------------------------------------------------------------
//Transformation: Create Service Endpoint
-------------------------------------------------------------------------------------------
@text-explicit
[template public CreateServiceEndpoint(service : Service, endpoint : InteractionPoint)]
        [let interface : ServiceInterface = service.realizedDescription.interface/]
        <service name = service.name interface = [interface.qualifiedName/]>
                <endpoint name = [service.name + "Endpoint"/]
                          binding = [endpoint.qualifiedName]
                          address = [endpoint.location/]/>
        </service>
[/template]
```

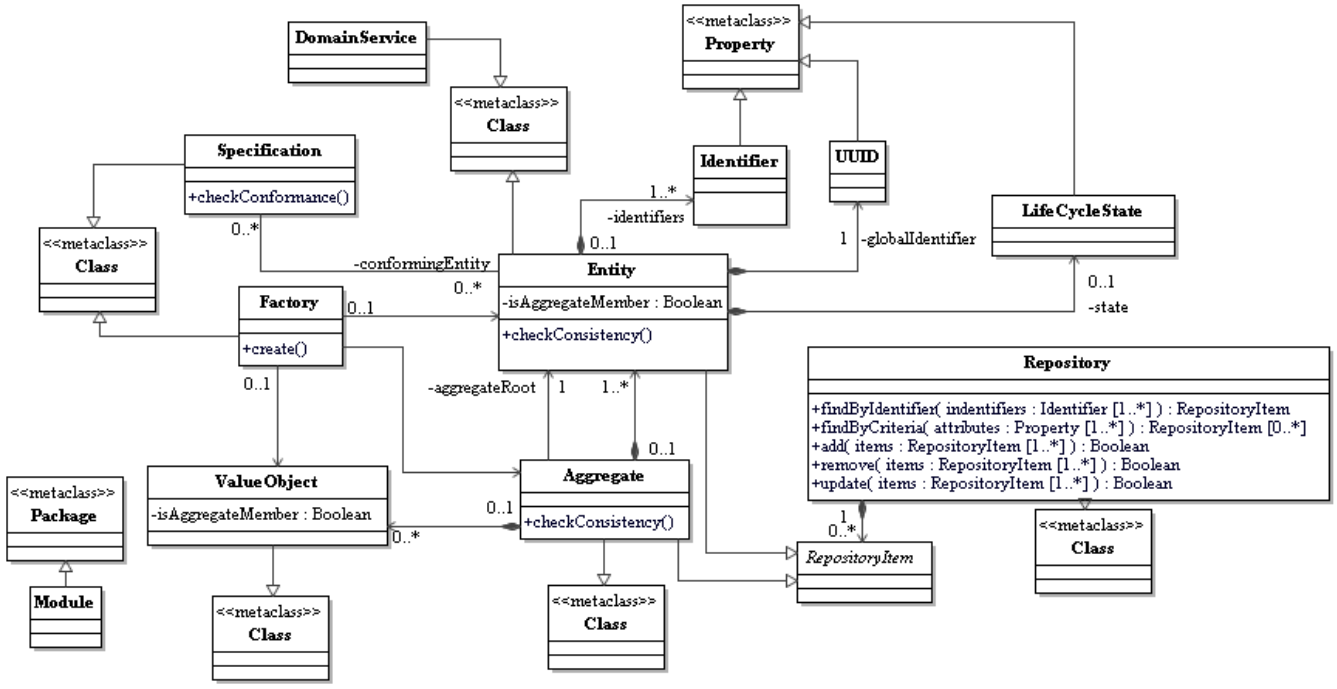
Listing 6: Create Service Endpoints

Appendix II

Evidence for Empirical Evaluation: Domain-Driven Design Metamodel to EMF Metamodels

Domain-Driven Design Metamodel

Step 1: Metamodel (using MagicDraw®)



Step 2: XMI Export of the Metamodel – Using XMI 2.1 Export

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xmi:XMI xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:MagicDrawProfile="http:///schemas/MagicDrawProfile/_UG7MoOetEd2dH_d60Sielg/0"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    xmlns:uml="http://www.eclipse.org/uml2/2.0.0/UML"
    xsi:schemaLocation="http:///schemas/MagicDrawProfile/_UG7MoOetEd2dH_d60Sielg/0
    UML_Standard_Profile.MagicDraw_Profile.profile.uml#_UG7NauetEd2dH_d60Sielg">
  - <uml:Model xmi:id="eee_1045467100313_135436_1" name="DomainDrivenDesign"
      viewpoint="">
      <ownedComment xmi:id="_15_0_1_6410187_1211888281480_158565_98"
        body="Author:I034949. Created:5/27/08 5:08 PM. Title:. Comment:."
        annotatedElement="eee_1045467100313_135436_1" />
    + <packageImport xmi:id="_0primitiveTypesDomainDrivenDesign">
    + <packageImport xmi:id="_0javaPrimitiveTypesDomainDrivenDesign">
    + <packagedElement xmi:type="uml:Class"
        xmi:id="_15_0_1_6410187_1211888318993_265824_254" name="Entity">
    + <packagedElement xmi:type="uml:Class"
        xmi:id="_15_0_1_6410187_1211888472890_35449_275" name="ValueObject">
    + <packagedElement xmi:type="uml:Class"
        xmi:id="_15_0_1_6410187_1211888499124_303717_296" name="Repository">
    + <packagedElement xmi:type="uml:Class"
        xmi:id="_15_0_1_6410187_1211888577402_844542_318" name="Identifier">
    + <packagedElement xmi:type="uml:Association"
        xmi:id="_15_0_1_6410187_1211888707851_323608_397" name=""
        memberEnd="_15_0_1_6410187_1211888707851_378243_399
        _15_0_1_6410187_1211888707851_992470_398">
    + <packagedElement xmi:type="uml:Class"
        xmi:id="_15_0_1_6410187_1211888891551_707739_452" name="Aggregate">
    + <packagedElement xmi:type="uml:Association"
        xmi:id="_15_0_1_6410187_1211889018939_974587_536" name=""
        memberEnd="_15_0_1_6410187_1211889018939_241627_537
        _15_0_1_6410187_1211889018939_195167_538">
    + <packagedElement xmi:type="uml:Association"
        xmi:id="_15_0_1_6410187_1211889117296_969811_593" name=""
        memberEnd="_15_0_1_6410187_1211889117296_588819_595
        _15_0_1_6410187_1211889117296_248004_594">
    + <packagedElement xmi:type="uml:Association"
        xmi:id="_15_0_1_6410187_1211889462119_724336_659" name=""
        memberEnd="_15_0_1_6410187_1211889462119_377407_660
        _15_0_1_6410187_1211889462119_408195_661">
```

```
+ <packagedElement xmi:type="uml:Class"
    xmi:id="_15_0_1_6410187_1211950868193_207254_403" name="Factory">
+ <packagedElement xmi:type="uml:Association"
    xmi:id="_15_0_1_6410187_1211951360311_43111_783" name=""
    memberEnd="_15_0_1_6410187_1211951360311_671053_785
    _15_0_1_6410187_1211951360311_378141_784">
+ <packagedElement xmi:type="uml:Association"
    xmi:id="_15_0_1_6410187_1211951870125_956542_890" name=""
    memberEnd="_15_0_1_6410187_1211951870125_331130_892
    _15_0_1_6410187_1211951870125_441816_891">
+ <packagedElement xmi:type="uml:Class"
    xmi:id="_15_0_1_6410187_1211951921860_257327_943" name="Specification">
  <packagedElement xmi:type="uml:Association"
    xmi:id="_15_0_1_6410187_1211951950752_630672_966" name=""
    memberEnd="_15_0_1_6410187_1211951950752_626150_967
    _15_0_1_6410187_1211951950752_875256_968" />
+ <packagedElement xmi:type="uml:Association"
    xmi:id="_15_0_1_6410187_1211957735532_377601_1378" name=""
    memberEnd="_15_0_1_6410187_1211957735532_184178_1380
    _15_0_1_6410187_1211957735532_412680_1379">
  <packagedElement xmi:type="uml:Class"
    xmi:id="_15_0_1_6410187_1211957987870_739727_1433"
    name="RepositoryItem" isAbstract="true" />
+ <packagedElement xmi:type="uml:Association"
    xmi:id="_15_0_1_6410187_1211958079887_888353_1456" name=""
    memberEnd="_15_0_1_6410187_1211958079887_381738_1457
    _15_0_1_6410187_1211958079887_300394_1458">
+ <packagedElement xmi:type="uml:Class"
    xmi:id="_15_0_1_6410187_1211968311206_246242_1574" name="Module">
+ <packagedElement xmi:type="uml:Class"
    xmi:id="_15_0_1_6410187_1211968943007_374719_1940" name="Service">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1211972576583_81467_1986" name="Aggregate">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1211972600756_857115_1988" name=""
    memberEnd="_15_0_1_6410187_1211972600756_242802_1990
    _15_0_1_6410187_1211972600756_99443_1989">

 + <packagedElement xmi:type="uml:Extension"
     xmi:id="_15_0_1_6410187_1211972600756_857115_1988" name=""
     memberEnd="_15_0_1_6410187_1211972600756_242802_1990
     _15_0_1_6410187_1211972600756_99443_1989">
 + <packagedElement xmi:type="uml:Class"
     xmi:id="_15_0_1_6410187_1211973268847_572802_2153"
     name="LifeCycleState">
 + <packagedElement xmi:type="uml:Association"
     xmi:id="_15_0_1_6410187_1211973508275_929347_2271" name=""
     memberEnd="_15_0_1_6410187_1211973508275_169984_2273
     _15_0_1_6410187_1211973508275_386083_2272">
 + <packagedElement xmi:type="uml:Extension"
     xmi:id="_15_0_1_6410187_1211978832289_200705_2373" name=""
     memberEnd="_15_0_1_6410187_1211978832289_593600_2375
     _15_0_1_6410187_1211978832289_820040_2374">
```

```
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1211978822226_321028_2371" name="ValueObject">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1211978903773_301357_2410" name="Service">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1211978906461_866593_2412" name=""
    memberEnd="_15_0_1_6410187_1211978906461_209095_2414
    _15_0_1_6410187_1211978906461_108980_2413">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1211979604851_228209_2496" name="Repository">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1211979612773_177191_2498" name=""
    memberEnd="_15_0_1_6410187_1211979612773_710062_2500
    _15_0_1_6410187_1211979612773_586974_2499">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212039487660_578855_2540"
    name="RepositoryItem">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212039490612_873689_2542" name=""
    memberEnd="_15_0_1_6410187_1212039490612_936128_2543
    _15_0_1_6410187_1212039490612_199499_2544">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040248817_727745_2722" name="Factory">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040257051_789614_2724" name=""
    memberEnd="_15_0_1_6410187_1212040257051_352167_2725
    _15_0_1_6410187_1212040257051_642274_2726">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040343662_651450_2772" name="Create">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040349803_173966_2774" name=""
    memberEnd="_15_0_1_6410187_1212040349803_48976_2776
    _15_0_1_6410187_1212040349803_317680_2775">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040410242_519342_2797" name="Module">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040414805_877935_2799" name=""
    memberEnd="_15_0_1_6410187_1212040414805_461995_2801
    _15_0_1_6410187_1212040414805_960493_2800">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040443618_219395_2822" name="Specification">
```

```
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040443618_219395_2822" name="Specification">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040450852_871387_2824" name=""
    memberEnd="_15_0_1_6410187_1212040450852_835288_2826
    _15_0_1_6410187_1212040450852_218195_2825">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040506854_443987_2847" name="add">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040509713_389094_2849" name=""
    memberEnd="_15_0_1_6410187_1212040509713_954991_2851
    _15_0_1_6410187_1212040509713_846836_2850">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040531776_178389_2872" name="remove">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040536948_795973_2874" name=""
    memberEnd="_15_0_1_6410187_1212040536948_103428_2876
    _15_0_1_6410187_1212040536948_932506_2875">
+ <packagedElement xmi:type="uml:Stereotype"
    xmi:id="_15_0_1_6410187_1212040623216_562271_2897" name="find">
+ <packagedElement xmi:type="uml:Extension"
    xmi:id="_15_0_1_6410187_1212040628576_469031_2899" name=""
    memberEnd="_15_0_1_6410187_1212040628576_512378_2901
    _15_0_1_6410187_1212040628576_519962_2900">
+ <packagedElement xmi:type="uml:Package"
    xmi:id="magicdraw_uml_standard_profile_v_0001" name="UML Standard Profile">
+ <profileApplication
    xmi:id="_9_0_be00301_1108050582343_527400_10847profileApplicationDomainDriven
+ <profileApplication
    xmi:id="_11_5_be00301_1153310565718_226811_161profileApplicationDomainDrivenD
+ <profileApplication
    xmi:id="_be00301_1073394351331_445580_2profileApplicationDomainDrivenDesign">
+ <profileApplication
    xmi:id="_11_5_f720368_1159529670215_231387_1profileApplicationDomainDrivenDesi
</uml:Model>
<MagicDrawProfile:auxiliaryResource xmi:id="_UIaac-etEd2dH_d60Sielg"
  base_Element="magicdraw_uml_standard_profile_v_0001"
  base_Package="magicdraw_uml_standard_profile_v_0001" />
```

Step 3: XMI to ECore Conversion – Using the wf-uml2ecore-policy.oaw Cartridge

```
<?xml version="1.0"?>
<workflow>
<cartridge
      file="org/openarchitectureware/util/uml2ecore/uml2ecoreWorkflow.oaw"
    uml2ModelFile="md/DDD.uml"
    nsUriPrefix="http://www.fictitious.org/domainmodel"
    addNameAttribute="true"
    includedPackages="DDDMetamodel"
    resourcePerToplevelPackage="false"
    outputPath="src-gen" />
</workflow>
```

Step 4: ECore Model

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
name="DomainDrivenDesign"
    nsURI="http:///DomainDrivenDesign.ecore" nsPrefix="DomainDrivenDesign">
  <eClassifiers xsi:type="ecore:EClass" name="Entity"
eSuperTypes="uml.ecore#//Class #//RepositoryItem">
    <eOperations name="checkConformance" ordered="false" lowerBound="1"/>
    <eStructuralFeatures xsi:type="ecore:EReference" ordered="false"
lowerBound="1"
        eType="#//Identifier" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isAggregateMember"
ordered="false"
        lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="RepositoryItem"
abstract="true"/>
  <eClassifiers xsi:type="ecore:EClass" name="Identifier"
eSuperTypes="uml.ecore#//Property"/>
  <eClassifiers xsi:type="ecore:EClass" name="Specification"
eSuperTypes="uml.ecore#//Class"/>
  <eClassifiers xsi:type="ecore:EClass" name="ValueObject"
eSuperTypes="uml.ecore#//Class">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isAggregateMember"
ordered="false"
        lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Repository"
eSuperTypes="uml.ecore#//Class">
    <eOperations name="findByIdentifier" ordered="false" lowerBound="1"
eType="#//RepositoryItem">
      <eParameters name="indentifiers" ordered="false" lowerBound="1"
upperBound="-1"
```

```
          eType="#//Identifier"/>
    </eOperations>
    <eOperations name="findByCriteria" ordered="false" upperBound="-1"
eType="#//RepositoryItem">
      <eParameters name="attributes" ordered="false" lowerBound="1"
upperBound="-1"
          eType="ecore:EClass uml.ecore#//Property"/>
    </eOperations>
    <eOperations name="add" ordered="false" lowerBound="1"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
      <eParameters name="items" ordered="false" lowerBound="1" upperBound="-
1" eType="#//RepositoryItem"/>
    </eOperations>
    <eOperations name="remove" ordered="false" lowerBound="1"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
      <eParameters name="items" ordered="false" lowerBound="1" upperBound="-
1" eType="#//RepositoryItem"/>
    </eOperations>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_" ordered="false"
upperBound="-1"
        eType="#//RepositoryItem" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Aggregate"
eSuperTypes="uml.ecore#//Class #//RepositoryItem">
    <eOperations name="checkConsistency" ordered="false" lowerBound="1"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="aggregateRoot"
ordered="false"
        lowerBound="1" eType="#//Entity"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_" ordered="false"
upperBound="-1"
        eType="#//ValueObject" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Factory"
eSuperTypes="uml.ecore#//Class">
    <eOperations name="create" ordered="false" lowerBound="1"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_" ordered="false"
lowerBound="1"
        eType="#//Entity"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Module"
eSuperTypes="uml.ecore#//Namespace"/>
  <eClassifiers xsi:type="ecore:EClass" name="Service"
eSuperTypes="uml.ecore#//Class"/>
</ecore:EPackage>
```

List of Publications

Peer-Reviewed Journal Papers

- Harshavardhan Jegadeesan, Sundar Balasubramaniam: "An MOF2-based Services Metamodel", in Journal of Object Technology, vol. 7, no. 8, Nov-Dec 2008
- Harshavardhan Jegadeesan, Sundar Balasubramaniam: "A Model-Driven Approach to Service Policies ", in Journal of Object Technology, vol. 8, no. 3, Mar-Apr 2009 (to appear)
- Harshavardhan Jegadeesan, Sundar Balasubramaniam: “Service Flavors: Differentiating Service Offerings in a Services Marketplace”, communicated to the Journal of Webservices Research on January 22, 2008.

Peer-Reviewed Conference Papers

- Harshavardhan Jegadeesan, Sundar Balasubramaniam: "Differentiating Commoditized Services in a Services Marketplace ", in the 2008 IEEE Conference on Services Computing (SCC 2008), Honolulu, Hawaii, USA, July 8 – 11, 2008.
- Sundar Balasubramaniam, Harshavardhan Jegadeesan: "eThens - A Modular Framework for e-Governance", Proceedings of the International Conference on Politics and Information Systems, Technologies and Applications (PISTA 2004),Orlando, Florida, USA, July 2004.

Brief Biography of Candidate and Supervisor

Harshavardhan Jegadeesan currently works as a Product Development Lead in the Enterprise SOA team within the Business Suite organization in SAP Labs, India. Prior to this he was working with the Research & Breakthrough Innovation group on SAP® ByDesign®. He is also a guest faculty with BITS, Pilani teaching object-oriented analysis & design for the collaborative graduate program is software engineering. He holds a Masters in Software Systems from BITS, Pilani. His areas of interest include service-oriented architectures, enterprise systems and business process platforms.

Ramana Polavarapu, Ph.D currently works with the Services Science group in IBM Research. Prior to joining IBM Research, he worked in SAP Labs (Palo Alto and Bangalore) as a Platinum Developer. Ramana has around ten years of experience in software design, architecture and programming. Earlier, he was an Assistant Professor of Economics in the University of Colorado at Denver where he taught international trade, game theory and industrial organization. He holds a Ph.D from the University of California at Davis.