# ARCHITECTURE SPECIFICATION OF SERVICE-ORIENTED SYSTEMS THROUGH SEMANTIC WEB TECHNOLOGIES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

VELİ BİÇER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2007

Approval of the thesis:

**ARCHITECTURE SPECIFICATION OF SERVICE-ORIENTED SYSTEMS THROUGH SEMANTIC WEB TECHNOLOGIES**

submitted by **VELİ BİÇER** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University by**,

Prof. Dr. Canan Özgen ____________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay ____________________
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ali Doğru ____________________
Supervisor, **Computer Engineering Dept., METU**

**Examining Committee Members:**

Assoc. Prof. Dr. Halit Oğuztüzün ____________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ali Doğru ____________________
Computer Engineering Dept., METU

Asst. Prof. Dr. Tolga Can ____________________
Computer Engineering Dept., METU

Dr. Meltem Turhan Yöndem ____________________
Computer Engineering Dept., METU

Dr. Aysu Betin Can ____________________
Informatics Institute, METU

**Date:** 14 August 2007

|  |  |
|---|---|
| Name, Last name | : Veli Biçer |
| Signature | : |

# ABSTRACT

ARCHITECTURE SPECIFICATION OF SERVICE-ORIENTED SYSTEMS THROUGH SEMANTIC WEB TECHNOLOGIES

Biçer, Veli

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali Doğru

August 2007, 87 pages

This thesis presents a semantic-based modeling approach for describing Service-Oriented Architectures (SOA). Ontologies are utilized as a major representation mechanism for describing various elements available in the architecture. The methodology proposes an architecture specification mechanism to constuct a unified ontology that enables transition from design concerns to the modeling elements. A multi-level modeling is also achieved by employing Model-Driven Engineering (MDE) techniques to describe various models at different stages of the software architecture. This aims to organize service-oriented models within a number of architecture viewpoints in order to provide an architectural perspective for SOA. The use of ontologies for model specification also allows us to make use of ontology mapping to specify the transformation between different models. Additionally, we present a case study to demonstrate the proposed methodology on a real-world healthcare scenario.

Keywords: Service-Oriented Architecture, Model-Driven Engineering, Ontology

# ÖZ

SEMANTİK WEB TEKNOLOJİLERİ KULLANARAK SERVİS-YÖNELİMLİ SİSTEM MİMARİSİNİN BELİRLENMESİ

Biçer, Veli

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Dr. Ali Doğru

Agustos 2007, 87 sayfa

Bu tezde, servis-yönelimli sistemlerin geliştirilmesi için semantik tabanlı bir modelleme yaklaşımı sunulmaktadır. Mimaride bulunan farklı bileşenlerin gösterimi için ontolojilerden faydalanılmıştır. Uygulanan metod, mimarinin belirtilmesi için birleşik bir mimari ontolojisi yaratmayı öngörmektedir. Böylelikle tasarım hedeflerinden modelleme bileşenlerine bir geçiş yapabilme hedeflenmiştir. Ayrıca, çok katmanlı modellemede yapabilmek için model güdümlü mühendislik tekniklerinin de kullanılması sağlanmıştır. Bu sayede farklı katmanlardaki servis-yönelimli modeller değişik bakış açıları içinde organize edilip mimari bir yapı sunmaktadır. Ontolojilerinin model gösterimi için kullanımı sayesinde ontoloji dönüşüm tekniğinin model transformasyonu için kullanımı sağlanmıştır. Önerilen metod sağlık alanında bir gerçek hayat senaryosu ile örneklendirilmiştir.

Anahtar Kelimeler: Servis Yönelimli Mimari, Model Güdümlü Mühendislik, Ontoloji

To my family

# ACKNOWLEDGMENTS

I thank my advisor Assoc. Prof. Dr. Ali Doğru for his guidance, expertise, and patience throughout this research. With his support, this study has been an invaluable learning experience for me. It has really been an honor to work with this consummate professional.

It may be unusual to thank an institution; nevertheless I feel the need to thank Middle East Technical University for providing such an excellent environment for self-development. I am very grateful to the people who built this splendid institution and put their efforts as members over the years.

I would like to express my thanks to my colleagues at Tepe International Inc., Erdal Özkazanç, Nejat Serpen, Ömer Hotomaroğlu, Volkan Er, İlyas Hacıömeroğlu, Tolga Özdemir and Murat Güneş, for their help and support during my graduate study.

I also would like to express my special thanks to my friends Buğrahan Akçay, Cengiz Togay, Coşkun Anaç, Erkan Vural, Mustafa Diken, Şeref Arıkan and Zafer Serbest for their morale support.

Finally, I would like to express my deepest gratitude to all members of my family, especially my mother Gülten Biçer and my wife Kader Biçer, for their love, understanding, and patience.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| AMMA | : Atlas Model Management Architecture |
| ATL | : Atlas Transformation Language |
| BPEL | : Business Process Execution Language |
| BPMN | : Business Process Modeling Notation |
| CCOW | : Clinical Context Management Specification |
| CCR | : Continuity of Care Record |
| CDA | : Clinical Document Architecture |
| CIM | : Computation Independent Model |
| CPP/CPA | : Collaboration Protocol Profile/Agreement |
| DICOM | : Digital Imaging and Communications in Medicine |
| DSL | : Domain-Specific Language |
| EBBP | : Electronic Business XML Business Process |
| EHR | : Electronic Healthcare Record |
| ESB | : Enterprise Service Bus |
| HL7 | : Health Level Seven |
| IEEE | : Institute of Electrical and Electronics Engineer |
| IHE | : Integrating Healthcare Enterprises |
| ISM | : Implementation Specific Model |
| KM3 | : Kernel Meta-Meta Model |
| MDE | : Model-Driven Engineering |
| MDA | : Model-Driven Architecture |
| MOF | : Meta Object Facility |
| OMG | : Object Management Group |
| OWL | : Web Ontology Language |

| | |
|---|---|
| P2P | : Peer-to-Peer |
| PIM | : Platform Independent Model |
| PSM | : Platform Specific Model |
| QVT | : Query/View/Transformation |
| RDF | : Resource Description Framework |
| RDFS | : Resource Description Framework Schema |
| SOA | : Service-Oriented Architecture |
| SOAP | : Simple Object Access Protocol |
| SOC | : Service-Oriented Computing |
| SOSE | : Service-Oriented System Engineering |
| UDDI | : Universal Description, Discovery and Integration |
| URI | : Uniform Resource Identifier |
| W3C | : World Wide Web Consortium |
| WSDL | : Web Service Description Language |
| WS-CDL | : Web Service Choreography Description Language |
| WSFL | : Web Service Flow Language |
| WSML | : Web Service Modeling Language |
| WSMO | : Web Service Modeling Ontology |
| WSMX | : Web Service Modeling Execution Environment |
| XML | : Extensible Markup Language |

# CHAPTERS

# 1. INTRODUCTION

Service-oriented architecture (SOA) and Web service technologies are opening a new era in computing with the promise of a complete application architecture that can deliver its functions across World Wide Web by using the emerging standards and a distributed system model. It is the latest software trend built on the advances in software engineering following the way from monolithic applications into SOA, through structured, object-oriented and component-based software development. The first generation of SOA has already been taken place with the advent of basic service engineering mechanisms to implement Web services to issue requests to others and to enable publishing, description, and discovery these services[1]. A new generation is already underway in terms of semantic Web services brings a new vision on top of this basis in order to create more dynamic service-oriented frameworks and increase the level of automation[2].

As presented in Section 2, there are various standards and concepts introduced through service-orientation in order to better utilize different aspects of enterprise systems. This situation provides some benefits such as a clear separation of concerns in terms of service creation, provisioning, composition, and management, but presents a complexity in the integration of the service-oriented concepts with enterprise application architectures in a seamless way. It is mainly due to the fact that the existence of these various technologies with their own conceptual models necessitates an architectural view for the alignment of the models within the enterprise systems. Additionally, it is a more challenging task than the traditional software architecture design, since the technologies come with very distinctive promises such as self-adaptability, runtime service binding, dynamic composition and semantic mediation. These characteristics of SOA require a software engineering methodology that allows the enterprises to effectively analysis, design and deploy service-oriented systems.

To propose a powerful mechanism in an effort to respond to the need of defining SOA, we develop an architecture specification infrastructure. Our approach leverages models in different layers of SOA technology stack by utilizing ontologies for a common foundation. This also enables the collaboration of various techniques in architecture specification such as analyzing the domain, defining various viewpoints and specifying the elements of the architecture. A multi-level modeling is employed to enable the applicability of high-level models to a variety of technologies. Although a complete methodology is foreseen and defined, this is solely done to present a context for the main contribution of this thesis - that is ontology-based architecture specification.

## 1.1. Status of enterprise software systems

Software systems have become a major part in modern enterprises providing a number of opportunities to change the way we conduct business. Today, in almost every domain, software is a dependable component to optimize process and knowledge. This reliance on software makes it a core component of today's business environments and a major source of innovation and growth.

Despite the growing dependence, software engineering faces new challenges in software development and maintenance due to the scale and complexity of the software systems and ever-changing technological world. Because there is an increasing demand from the users for modern software systems to attain precise characteristics such as distribution, agility, reliability, sustainability, flexibility and security, the software engineering community deals with radical paradigm shifts to ensure the evolution and quality of the software and to integrate emerging technologies into enterprise systems. There is a strong correlation between the ability of an enterprise to manage the complexity of its information systems and its capability to adapt to rapid business changes. This enables the enterprises to pursue fundamental transformation in order to gain a competitive advantage [3].

In order to better cope with the complexity of software systems at enterprise level, there is a recent trend which originates from perceiving all resources as services. In this scope, SOA provides many opportunities as a major computation paradigm. It utilizes the services and related service-oriented technologies to offer primary enabling technology to realize interconnected systems working in a single information space. Although SOA has been introduced to cope with internal and external application integration, the potential benefits of SOA are even much bigger in software engineering. First, SOA has successfully achieved the idea of separation of concerns by successfully adopting different concepts from various areas of computer science. Increasingly, the notion of service-oriented systems marks another step away from rigid, monolithic systems towards highly flexible dynamic and adaptive systems.

However, the availability of the technology is not the only necessary factor to achieve the inventive potential of SOA. The real challenge actually lies in the effective use of the service-oriented technologies to assemble systems from a set of services and service-oriented middleware available in different contexts with minimum effort and maximum reusability as well as maintaining the agility, autonomy and interactivity. This is hard to achieve without a preceding strategy to take place for the alignment of the business requirements and the technologies offered by the platform [4].

## 1.2. Multitude of service-oriented technologies

SOA, in fact, depends on very basic principles: providing services through standardized interfaces and allowing the external applications to use the distributed functions of these services in a message-oriented way. However, the requirements of the modern software architectures such as service composition, semantic mediation, automatic discovery, runtime management/monitoring, security, reliable messaging, transactions, event notification etc. bring about the need to add many logical layers on top of the services. To provide these standardized complex functionalities, various service-oriented middleware platforms have been developed by open source software community [5-9] or commercial IT vendors. As these middleware platforms continue to change and evolve and enterprise application architectures tend to contain a broad mix of them, this causes a portfolio complexity in realizing service-oriented software systems [10].

By employing these logical layers with the corresponding middleware platforms, the service-oriented systems start with the idea of enabling a business-oriented design where the services mainly replaces the previous idea of business components [11] as coarse-granular software modules and service compositions enable the specification of the business processes on top of the services. However, it is at the enterprise level that most of the complexities of service-oriented system engineering occur since the service-orientation is not sufficient by itself to address many challenges of enterprise development. Over the recent years, service engineering community has proposed various approaches to align the business with the service-oriented architectures [12-15]. However, the research opportunities in service-oriented computing, as stated in [16], will continue to add many layers to the service-oriented technology stack in the future increasing the complexity for the enterprises in the adoption and implementation of service-oriented architectures. In order to realize an effective system development, service-orientation should be employed as an underlying distributed computing platform offering complex functionalities for enabling a design process. Existence of huge and enormously complex diversity within SOA concept presents a challenge to achieve this when applied to enterprise-wide service-oriented systems.

Therefore, through a particular software engineering methodology, the alignment of various service oriented models within enterprise system architecture is needed. This is mostly due to the fact that traditional software engineering paradigms for object-oriented and component-based development cannot be straightforwardly applied to service-oriented systems due to the very distinctive characteristics of SOA [16]. In addition, the design of service-oriented software architectures is a more challenging task than the traditional software architecture design, since SOA technologies come with the promises such as composition, ad-hoc service binding, adaptability or semantic mediation.

## 1.3. Related Concepts

Recent advances in software engineering built around the Model-Driven Engineering (MDE) concept provide a complementary approach to SOA that can facilitate the development of service-oriented systems. By utilizing MDE, it is possible to enable model-centric service-oriented system development by introducing various service-oriented technologies as underlying platform models in the architecture. This differs from the general-purpose modeling approaches to service-oriented system engineering, as presented in [17, 18], which provides a black-box modeling approach exposing only a top-level model with the particular model compilers or tools. On the other hand, the use of MDE enables a methodology progressing from highly specialized domain-specific models to the service-oriented technology models. This is, in fact, a multi-level modeling approach in which each level involves different kinds of models at different stages of abstraction and the transition between the levels are handled with model transformations [19]. This aims to allow a better alignment of business requirements, domain-specific abstractions and constraints with various service-oriented technologies.

In addition, a particular SOA implementation to be specified as a set of models is also distinguished by the features it supports, where a feature is an increment in system functionality. Feature Models [20] are the mostly used technique to identify the functional capabilities based on the problem domain within a hierarchy. Through feature modeling, we are able to capture the domain requirements as features that are organized and classified in a formal way. The features also drive the further steps in architecture specification by acting as a common ground between the problem and the solution domains.

Finally, to organize this variety of information that is essential to service-oriented system design, the previous achievements in software architecture field can be employed. Over the last decade, software architecture has emerged as a way to specify the overall system architecture by shifting the focus away from the implementation issues towards a more abstract level of detail. According to [21], we can define the software architecture as "fundamental organization of a system, embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution”. As the importance of the architectural design increases over the years, the approaches to software architecture has changed from the traditional component and connector approach to the composition of a set of architectural design decisions [22]. This requires the architectural decisions to be captured as first-class representations with clear semantics in addition to the documentation as proposed in [23] and [24].

## 1.4. Utilizing ontologies for architecture specification of SOA

In order to achieve an architecture specification, we propose a semantic-based modeling framework for SOAs. Ontologies play a key role in the proposed methodology in order to describe the various stages of the service-oriented systems engineering. They allow the system developers to capture a range of service-oriented and domain-specific models and place them into an architecture specification. Additionally, system features and other architectural elements can be specified by the ontologies with their unique relationships and semantics.

By employing ontologies and related semantic Web technologies as an underlying infrastructure, we have developed the following methodology for this thesis work:

- A two-phase methodology is proposed to develop SOAs by separating the architecture specification and application engineering processes. Since SOA involves a number of technologies and platforms to be precisely aligned prior to application engineering, architecture specification is a required step to construct systems at an enterprise-level.
- An architecture specification ontology is provided by extending the all-purpose IEEE standard[21]. With the purpose of enabling a transition from stakeholders’ concerns to the service-oriented models, it is supplied with the required references to the other specifications such as features and MDE elements.
- A feature model ontology is provided to represent the functional capabilities of the SOA in consideration. The features in the model are also

referred from the architecture ontology as the elements that bridge the stakeholders' concerns to one or more architectural viewpoints.

- A model specification ontology is developed based on the previous achievements in MDE to represent any service-oriented and domain-specific model in the architecture specification. A way of managing the models in an ontology repository is also demonstrated.
- Ontology mapping is employed in the architecture to enable the transformation of our ontology-based models. The mapping definitions together with the model specifications are also encapsulated in the architecture within the viewpoints which can be regarded as basic building blocks of SOAs.

## 1.5. Outline of the Thesis

In this chapter we provide a brief introduction to service-oriented system development. In Chapter 2, we focus on the advances in SOA by introducing the common standards and technologies available in service-oriented technology stack. In Chapter 3, we give the idea of MDE and present the current approaches to MDE. Semantic Web vision and technologies are also presented in Chapter 4 as background information. In Chapter 5, we present the details of the proposed approach by stating a two-phase methodology, namely platform engineering and application engineering. The details of the methodology are given with the clear motivations and illustrations. A case study is conducted in Chapter 6 in order to show the use of the methodology and present an evaluation based on scenario in a healthcare domain. Finally, in Chapter 7, we present our conclusions and address directions for possible future work.

# 2. SERVICE-ORIENTED COMPUTING

As the importance of Web and distributed applications increase, Service-Oriented Computing (SOC) has emerged as a new approach that uses the services and service-related technologies to realize distributed and flexible architectures in Web-enabled settings. A key outcome of this approach is expected to be the capability to develop applications in a timely and interoperable way by overcoming the heterogeneity-related difficulties of platforms. Services are the basic entities in SOC as autonomous software components which can be described, published, discovered and dynamically assembled to perform business activities and transactions. Since services are made available in a way that is independent of any platform, loose coupling is achieved between the service provider and service user enabling more flexibility in software systems.[1, 16, 25]

In a broad sense, SOC covers a range of subjects that find their origins in various disciplines of computer science. These subjects are merged together in sophisticated ways to find potential solutions to the increasing demands of modern enterprises[2], to open new perspectives in today dynamic business environments[26], and to boost ongoing innovation and research[16]. This section provides the necessary background about many concepts and technologies that coexist within the scope of SOC, today.

## 2.1. Foundations of Service-Oriented Computing

As the distributed computing gains more attention, the drawbacks of the distributed object-oriented systems become more apparent in terms of handling latency, partial failures and concurrency and the lack of shared memory access[27]. This means that distributed systems comprise unique characteristics different from local computing which make the vision of unified objects of 1990s very difficult to achieve. On the contrary, Service-Oriented Architecture supported with Web service technologies are proposed as a new distributed system architecture to address these characteristics such as message orientation, self-description, platform-neutrality, or network orientation [28].

The most typical and basic scenario for service-orientation is the one depicted in Figure 1. Here, the service provider defines a service-description to be published to the service registry. This description is then discovered by the service consumer; used to bind with the provider; and to interact with the service. The service registry, consumer and provider are all software agents providing or using some services. The interaction takes place as an exchange of the messages among these agents.

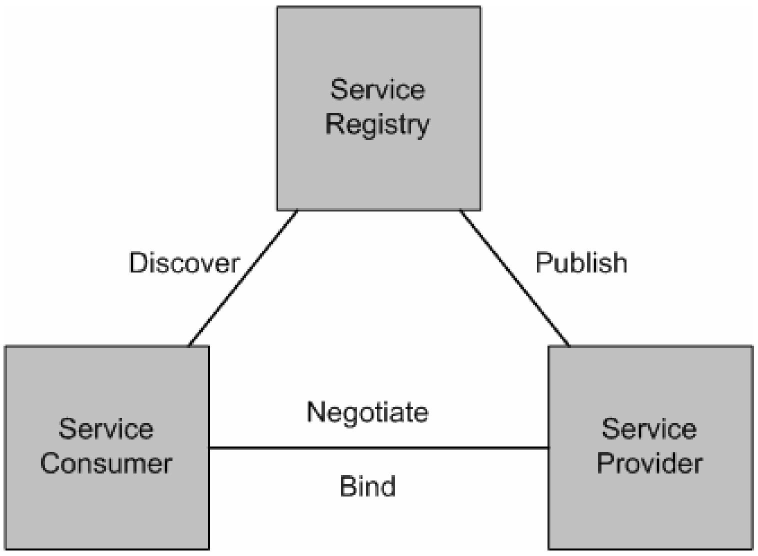


**Figure 1 The general service interaction pattern**

The most promising example to realize service-oriented architectures, today, is the Web services technologies. They rely on open Web standards such as Extensible Markup Language (XML), Simple Object Access Protocol (SOAP) or Web Service Description Language (WSDL). These standards enable the enterprise software to be developed in a distributed fashion with different tools and platforms supplied by different vendors- joining the software modules in different organizational departments or, possibly, outside the enterprise [25]. Additionally, the services to perform more complex business transactions can be realized, possibly on the fly, through the combination or configuration of the available services.

The essential Web Service specification and description depends on three basic standards: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). SOAP [29] is an XML-based communication protocol for exchanging information between agents apart from their platforms. It is defined as a lightweight protocol for exchange of structured and typed information in a decentralized, distributed environment. It includes three main parts: an envelope to specify what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; but, mainly used in combination with HTTP protocol to transport XML-encoded data.

WSDL [30] is a service description format used to describe the details of the service interfaces in a machine-processable XML format. It specifies the operations, messages, and a set of endpoints operating on messages to bind to a concrete network protocol. Although it is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, it is mainly used in conjunction with SOAP 1.1, or HTTP GET/POST. WSDL acts as a contract between the service provider and consumer to inform the consumer agent on how to invoke a service and what the data to be exchanged is.

UDDI [31] is another description protocol for distributed Web-based information registries of Web services - essentially repositories that store information about available web services. Service Providers can register information about the Web services they offer with these registries, and this information can then be discovered and accessed by Service Requestors. A key concept within UDDI is UDDI business registration - an XML based file used to describe a business entity and its Web services. Information captured within this file includes contact based information (business address, identifiers etc), categorizations of the business and services (using taxonomies, etc), and technical information (specifications of the services, etc). This information can then be used to help service requestors locate relevant services.

Although these basic standards and specifications are sufficient to realize a simple scenario as shown in Figure 1, the requirements of the modern software systems can be more demanding. For instance, the mediation of the exchanged messages, management of the metadata about agents and services, dynamic discovery, adaptation and composition are just some examples required to build more complex distributed application architectures. Recently, a number of middleware technologies and platforms have been developed to enable these complex functionalities. Enterprise Service Bus (ESB) is one of these middleware technologies aiming to provide an integration layer inserted between the service consumer and other agents. ESB is an open-standards based message backbone with a set of infrastructure capabilities to promote loose coupling of the systems and to break up the integration logic into distinct manageable pieces [32].

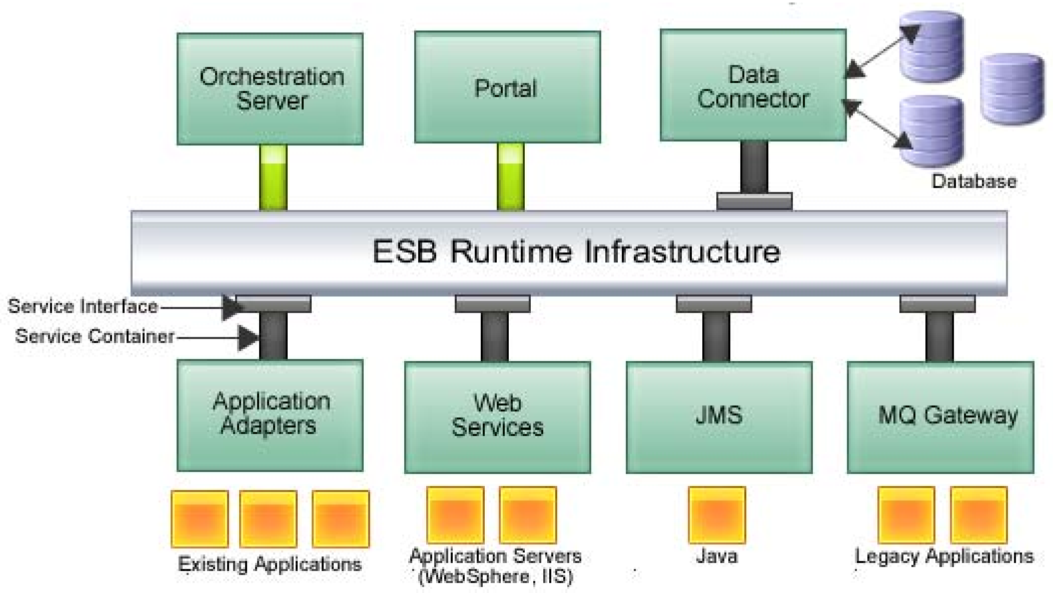


**Figure 2 Overview of Enterprise Service Bus connecting various platforms**

Figure 2 shows a simplified view of an ESB that integrates a various applications and technologies through a messaging backbone. In addition to the Web services developed in different platforms, ESB architecture allows the integration of the other applications such as Java/C# applications, mainframes, databases or portals. These applications are integrated into ESB through service containers and interfaces. A service container is a managed environment that hosts, manages, dynamically deploys services and binds them to external resources, e.g., data sources, enterprise and multi-platform applications.

## 2.2. Service Composition

The basic specifications of SOA enable the essential capabilities such as description, discovery and communication. Service composition provides a higher-level layer on top of these capabilities to allow the developers to aggregate multiple services into a composite added value services and enables the specification and management of integrating business processes [25]. This is an important functionality in creating service-oriented systems since the developers can easily create new services by combining the available basic services in order to enable integration through business processes that span organization boundaries. This process introduces a new role in service-oriented computing, namely service aggregator [32], which acts both as a service consumer and service provider to provide the combined functionality. This is depicted in Figure 3.

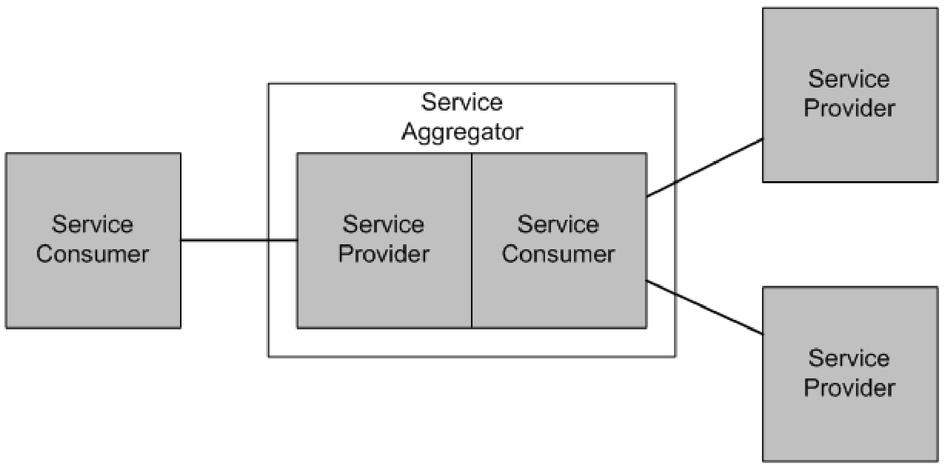


**Figure 3 Service Aggregator**

A number of specifications have emerged for service composition based on Web service technologies, after the core Web service capabilities are fully developed. These specifications provide different ways to define service compositions with their own meta-models and notations. *Choreography* and *orchestration* are introduced as two terms that describe and categorize the service composition specifications according to their very distinctive characteristics [15, 33]. Choreography specifies the interactions among multiple parties together with the message exchanges, rules of interaction and conditions. The coordination is distributed in choreography differing from orchestration, which coordinates the sequence of the interactions from the perspective of a single party based on a central control. Choreography provides a global view of the coordination of the service composition whereas orchestration specifies executable business logic which is defined as a long-lived, transactional process [34]. An illustration of choreography and orchestration is shown in Figure 4. Orchestration is mainly addressed by the languages such as Web Service Flow Language (WSFL) [35] and XLANG [36], but they are, then combined into a most representative language, namely Business Process Execution Language (BPEL) [37]. In addition, a number of standards addresses the choreography specification such as Web Services Choreography Description Language (WS-CDL) [38], Business Process Modeling Notation (BPMN) and ebXML Business Process (ebBP) [39] specification. In this section, we give an overview of these widely accepted service composition specifications.

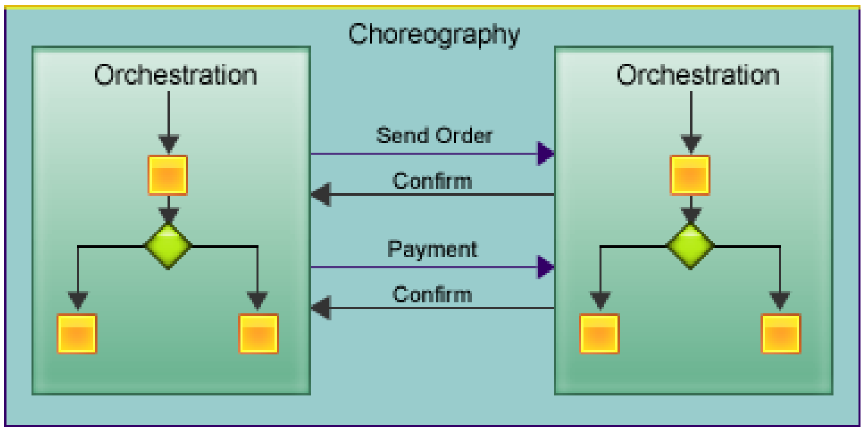


**Figure 4 Choreography and Orchestration**

### *2.2.1. Business Process Execution Language (BPEL)*

BPEL [37] is an XML-based language to describe orchestrations as a sequence of Web service interactions from a single party's point of view. It is built upon the other Web service standards such as WSDL, SOAP, XML Schema and XPath. It defines a model and grammar for specifying an executable business process based on interactions between the party and its partners. These interactions are coordinated to achieve a particular business goal with the specified operations and business logic in the process.

A composition defined in BPEL is represented with the *process* element including several element groups such as:

- Partner links: definition of a relationship with a partner by defining the message and port types used in the interactions in both directions.
- Variables and intermediate data operations: Variables are the way of managing the state of the business process by holding messages exchanged with the partners. Various tasks can also be performed on variables such as *assign*, or *copy* within the logic of the business process.
- Structural Activities: These activities define the order in which the basic activities occur. They include the sequential control activities such as

*sequence*, *switch* and *while* as well as the concurrency and synchronization with *flow* and event-handling with *pick*.

- Call activities: These are the activities to use (i.e. *invoke*) a Web service provided by a partner or to provide a Web service operation with *receive* and *reply* activities.
- Error handling: Errors are handled by using the fault handlers to catch and deal with faults and the compensation handlers used to undo already completed activities.

### *2.2.2. Web Services Choreography Description Language (WS-CDL)*

WS-CDL [38] is an XML-based choreography language that describes cross-enterprise collaborations of Web Services by defining their publicly visible behavior. It provides a high level coordination layer by filling the choreography gap that BPEL does not support. This enables the means to describe the collaborations precisely by specifying the rules of engagements among the partners. A WS-CDL model involves the following entities to specify the choreographies:

- Role types, relationship types and participant types: These types describe how the parties are capable of engaging in collaborations. The participants are abstracted by the participant types and the observable behaviors exhibited by the participant are represented as role types. All interactions that occur between roles are also constrained by relationship types.
- Information type, variable and token: A variable contains information about commonly observable objects in collaboration, such as the information exchanged or the observable information of the role types involved. A token is an alias that can be used to reference parts of a variable. Information exchange variables, state capturing variables and tokens have information types that define the type of information the variable contains or the token references

- Choreography and choreography life-line: Choreography defines collaborations between interacting participant types. The choreography life-line expresses the progression of these collaborations.
- Channel type: A channel realizes a point of collaboration between participant types by specifying where and how information is exchanged.
- Activities and ordering structures: Activities describe the actions performed within choreography. Ordering structures combine activities with other ordering structures in a nested structure to express the ordering rules of actions performed within choreography.

### *2.2.3. ebXML Business Processes (ebBP)*

ebBP [39] specification aims to describe choreographies among different partners based on the ebXML architecture. It facilitates the business processes as follows:

- In order for enterprises to collaborate with each other, they must first discover each other and the products and services they offer. ebXML provides a registry/repository architecture specification where such information can be published and discovered. A repository is a location (or a set of distributed locations) where a document pointed at by the registry resides and can be retrieved by conventional means (e.g., http or ftp). The repository is capable of storing any type of electronic content, while the registry is capable of storing metadata that describes content. The content within the repository is referred as “repository items” while the metadata within the registry is referred as “registry objects”.
- An enterprise needs to determine which business processes and documents are necessary to communicate with a potential partner. Registry metadata can be used for searching relevant documents and business processes. A Collaboration Protocol Profile (CPP) provides the details of how an organization is able to conduct business electronically. It specifies such items as how to locate contact and other information about the organization, the types of network and file transport protocols it uses,

network addresses, security implementations, and how it does business by providing a reference to a Business Process Specification. A Business Process Specification Schema (ebBP) in ebXML provides the definition of an XML document that describes how an organization conducts its business. While the CPA/CPP deals with the technical aspects of how to conduct business electronically, the ebBP deals with the actual business process.

- After the enterprises discover each other, they need to determine how to exchange information. The Collaboration Protocol Agreement (CPA) specifies the details of how two organizations have agreed to conduct electronic business. It is formed by combining the CPPs of the two organizations.

## 2.3. Service-Oriented Management

Management in a SOA environment is a required functionality in order to better monitor and utilize the distributed resources and facilitate the management tasks. Unlike the conventional computing, in which the management operation mostly deals with the operation of the hardware resources, service-oriented management enables the management of various distributed information technology resources ranging from services, service platforms and business processes to the autonomous systems as stated in the vision of autonomic computing [40]. Therefore, this is a more complex task since it considers various factors such as economic activities, failure detection, service-level agreements, capacity planning and policy matchmaking [2].

A generic conceptual architecture for the service-oriented management is introduced in [41] as shown in Figure 5. Each managed resource in the architecture is exposed by a management interface which provides the required operations, properties and events to the management applications. Managed resource is supplied with the metadata and other support mechanism in order to specify the resource properties and relationships and provide APIs for performing various tasks. The interface is used by the resource manager for performing management operations such as monitoring, analyzing planning and execution.

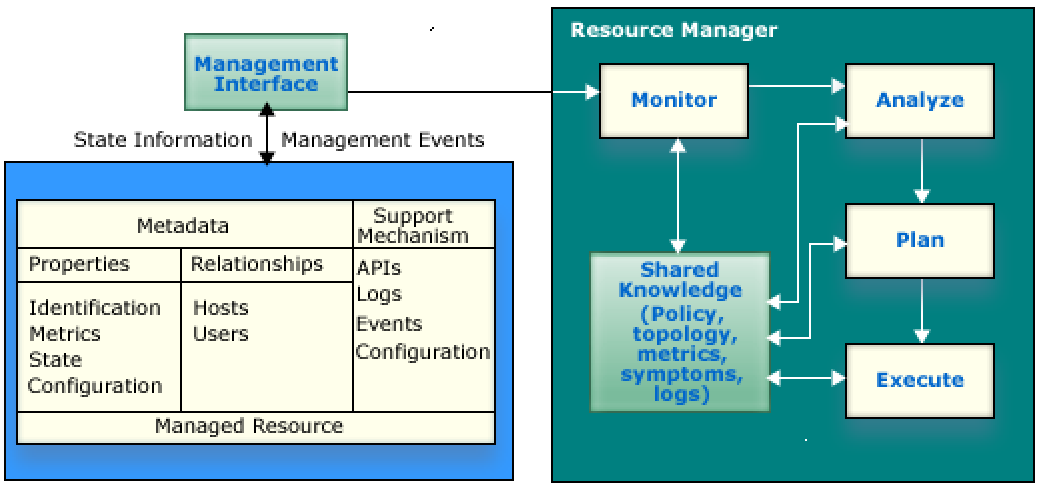


**Figure 5 A Conceptual Architecture for Service-Oriented Management**

This conceptual architecture is addressed by many specifications currently in use. Web Services Management Framework (WSMF) [42] is one of these specifications aiming to provide a logical architecture for the management of resources through Web services. A managed object in WSMF provides a set of management capabilities by implementing management interfaces described using WSDL. Therefore, a managed object provides a management Web service to enable a number of management functions, including:

- Discovery of the management Web service descriptions
- Discovery of the capabilities and event notifications
- Subscription to the events and notifications
- Expose additional management operations for six core categories such as monitoring, discovery, control, performance, configuration, and security

The use of Web service technologies in WSMF offers a platform-neutral model for management. WSMF provides the required data type schemas, WSDL templates and guidelines for describing manageability information (WSMF-Foundation), rules for advertising, subscribing, producing and consuming events (WS-Events), and an execution environment architecture to perform management (WS-Management).

Web Services Distributed Management (WSDM) is another specification which aims to combine service management and application channels developed in accordance with SOA principles. It delivers two sets of specifications: *Management Using Web Services (MUWS)* addresses the use of Web services technologies as the foundation of a modern distributed systems management framework to facilitate interactions between managed resources and management applications. *Management of Web Services (MOWS)*, on the other hand, addresses the specific requirements for managing Web services themselves. In WSDM, Web services are the platform for providing essential distributed computing functionality, interoperability, loose coupling, and implementation independence. The MOWS specification is mainly based on the MUWS specification's concepts and definitions.

Recently, the WSDM specification is mostly used to realize the vision of autonomic computing [43]. It mainly provides a solid base to specify the *touchpoints* which are autonomic computing system building blocks implementing sensors and effectors and exposing them through a manageability interface. Autonomic computing architecture extends this by building a autonomic manager which consists of one or more control loops to dynamically manage various aspects of a computing infrastructure. This is illustrated in Figure 6.

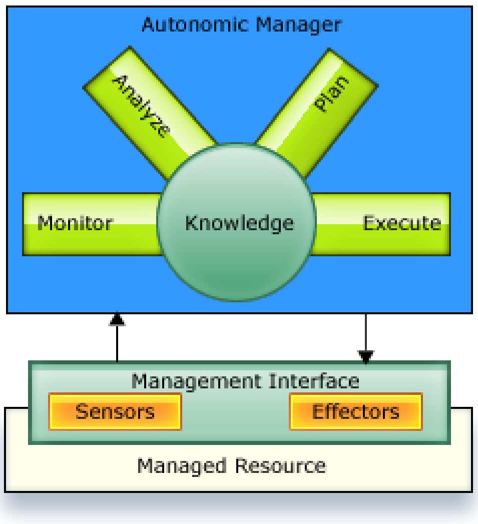


**Figure 6 Basic Autonomic Computing Architecture**

## 2.4. Service-Oriented System Engineering

The primary aim of service-oriented computing is to utilize services and the related technologies in order to support the development of low-cost, flexible, distributed and business oriented software systems. Beside some uses of Web service technologies for application integration at inter-enterprise level, the real potential of the service-oriented computing is enabled when the enterprises effectively build software architectures by using the emerging service-oriented concepts and platforms [16]. Service-Oriented System Engineering (SOSE) is concerned with methodologies and tools to build enterprise applications through employment and coordination of loosely-coupled, distributed services and available service-oriented concepts, standards and middleware. As service-oriented technologies gain significant acceptance by software industry, this becomes a critical issue in order to productively design large systems, and profit from the benefits of service-orientation.

As any other software system, a service-oriented system requires an architectural design. This is mainly due to the fact that the complexity of the service-oriented systems can be huge considering the very distinctive characteristics of the service-oriented computing such as the number of existing standards and middleware platforms, distributed nature of execution context, transaction management, inter-organizational security and trust, service provisioning, composition, discovery and coordination. A generic conceptual framework has been provided by [21] as a formal standard to address architectural description of software-intensive systems. It establishes the following main goals for the architectural design:

- Introducing the various stakeholders of the system, each with specific concerns (i.e. functionality, security, performance or reliability),
- Providing the architectural descriptions which contain particular design artifacts and architectural views,
- Formalizing the views of the system, each of which address one or more concerns of the system stakeholders,
- Linking each view to a viewpoint in order to establish the conventions and determine the languages, models, modeling methods and analysis techniques.

In the conceptual model as depicted in Figure 7, each system has an architecture which is described by an architectural description. The system also evolves in an environment and fulfils a specific mission. The architectural description contains views and models for the organization of the system.

The concepts of view and viewpoint are very crucial in this model since the system actors analyze the problem and solution domains by considering many viewpoints. This process is also called viewpoint hopping in which different subjects at different level of abstractions are explored during system analysis and design [44]. However, IEEE Std 1471-2000 does not specify any fixed set of viewpoints but provides a reference to define them.

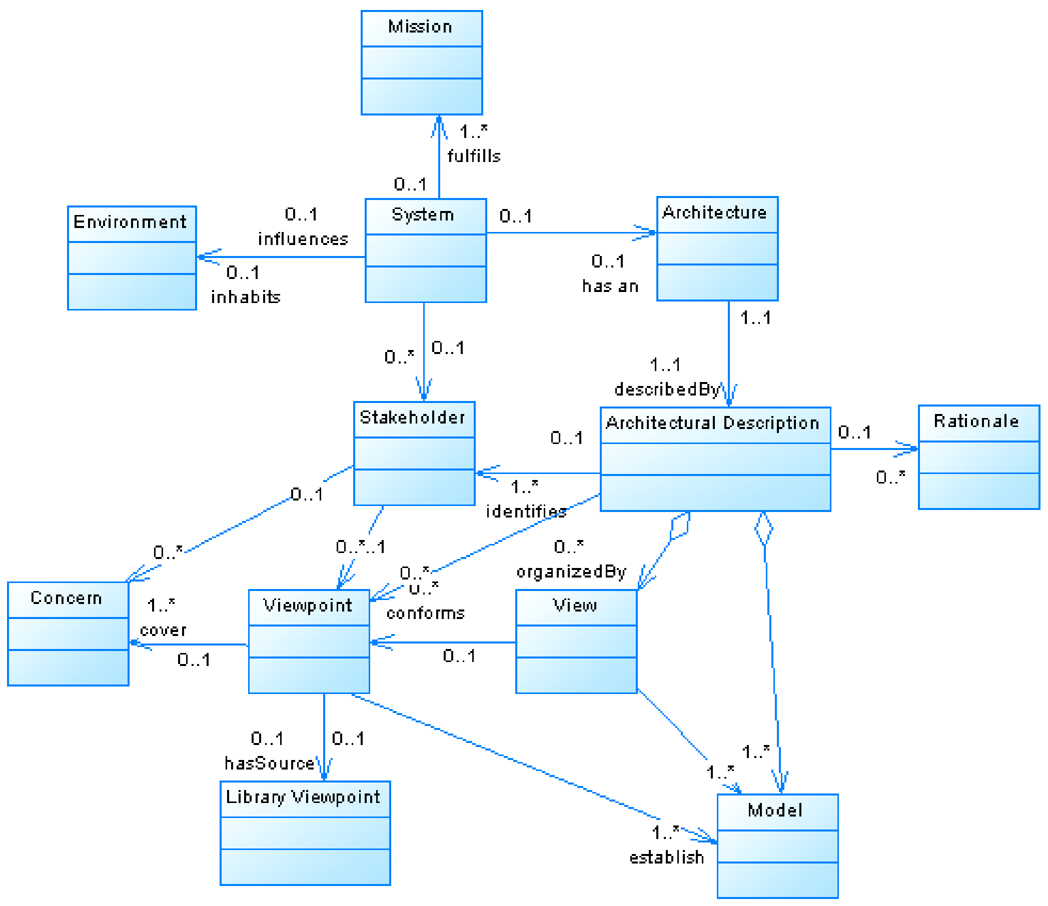


**Figure 7 IEEE Std 1471-2000 Conceptual Model [21]**

Service-oriented architecture, on the other hand, classifies a number of viewpoints available during design process. Although these viewpoints are closely related with each other, their definitions differ according to scope, context and the concerns of the actors in system engineering process. Based on the technologies and concepts presented in this chapter, we identify the following viewpoints:

- *Service Component Viewpoint*
- *Service Data Viewpoint*
- *Service Choreography Viewpoint*
- *Service Orchestration Viewpoint*
- *Service Semantics Viewpoint*
- *Quality of Service Viewpoint*

- *Service Management Viewpoint*

The activity of service-oriented system engineering utilizes these viewpoints to offer mechanisms for the system integrators to perform the functional stages of the system development. For design and development of the system, these stages involve the service specification, composition, discovery and testing. For the deployment, this involves the publication of the services. During runtime, the execution, management and adaptation stages can be performed.

# 3. MODEL-DRIVEN ENGINEERING

With the aim of improving the productivity, assuring the quality and reliability, and better managing complexity, Model-Driven Engineering (MDE) has emerged as a new software development paradigm that leverages many approaches in a synergistic way to meet the requirements in the development of modern software systems. MDE uses the models and model technologies in order to provide the level of abstraction for the system architects and developers to create software in a simplified and standardized way [45]. This level of abstraction also leads to the separation of concerns from business neutral descriptions and technology specific implementations by expressing specific aspects of the system under development as a set of models.

The basic entity in MDE is the *model*. A model is an abstraction specifying a certain aspect of the system by formalizing the entities and the relationships in a well-defined modeling language. A *modeling language* is used to express the models with its well-defined, abstract syntax. This syntax, which is also represented as a model, is called the *meta-model* of the model. In this respect, each model conforms to its meta-model for its formal definition. Similarly, the meta-models also conform to meta-meta-model which provides generic abstractions and syntax for defining meta-models. A meta-meta-model can be stated as a model which is its own reference model (i.e. it conforms to itself) [46]. This model specification hierarchy is depicted in Figure 8.

In addition to the models, model transformations play a key role in MDE in order to generate new or changed models from existing ones increasing the productivity and decreasing development time. With the help of the meta-modeling technique, the abstract syntax and semantics of the source and target models are clearly defined, which is one of initial requirements to generalize a model transformation approach. Based on the meta-models, a model transformation can be specified to automate mapping of all source models to the target models. Although the idea of transformation is not relatively new in computer science and software engineering, MDE proposes a more generic and automatic approach supported with general patterns and tools [47].

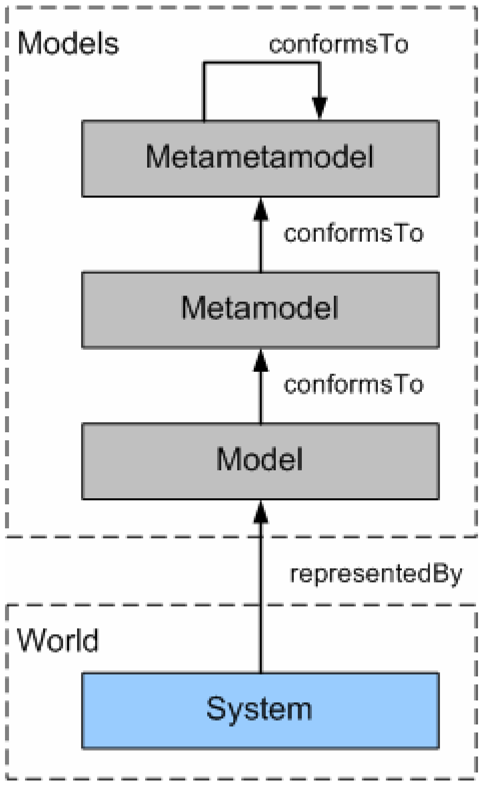


**Figure 8 Definition of Models**

## 3.1. Model Driven Architecture

A typical realization of MDE is provided by Object Management Group (OMG) in its Model Driven Architecture (MDA) specification [48]. MDA is defined to have a set of layers and transformations that provide a conceptual framework and vocabulary for system development. There are four kinds of models as depicted in Figure 9:

- *Computation Independent Model (CIM)* is used to specify the domain model of the problem domain from a computation independent viewpoint.

- *Platform Independent Model (PIM)* specifies a view of the system in a technology-neutral way.
- *Platform Specific Model (PSM)* represents the system with the details and mechanisms of particular implementation platform.
- *Implementation Specific Model (ISM)* specifies the implementation of the system using a particular programming environment and tools.

In order to introduce the models in MDA through their meta-models, Meta-Object Facility (MOF) standard of OMG is used as a meta-meta model. The MOF specification is used to model itself as well as other meta-models. It specifies the shared structure, and semantics of models in a concrete syntax based on XMI. Additionally, the new 2.0 version includes additional capabilities defined in separate packages including support for identifiers, additional primitive types, reflection, and simple extensibility through name-value pairs.

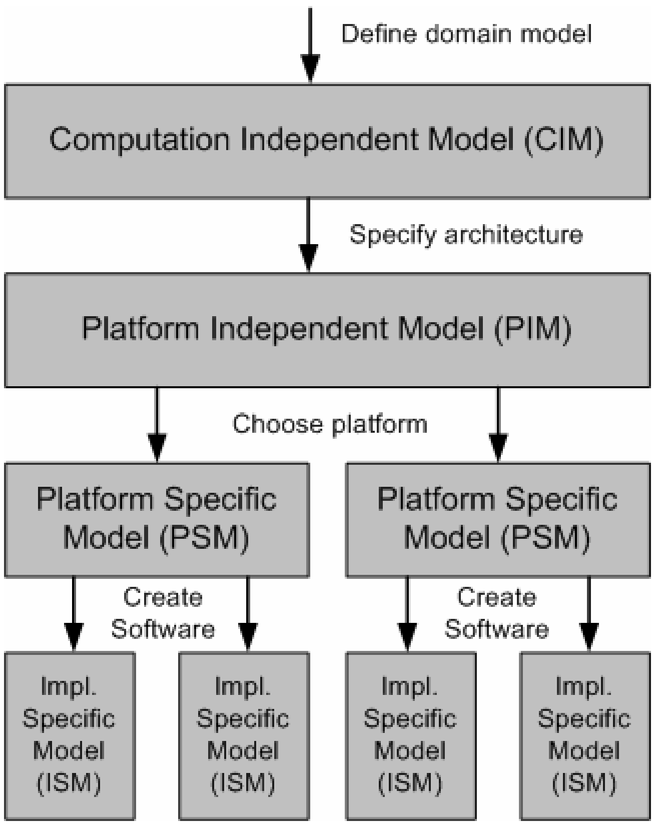

Define domain model
Computation Independent Model (CIM)
Specify architecture
Platform Independent Model (PIM)
Choose platform
Platform Specific Model (PSM)
Platform Specific Model (PSM)
Create Software
Create Software
Impl. Specific Model (ISM)
Impl. Specific Model (ISM)
Impl. Specific Model (ISM)
Impl. Specific Model (ISM)

**Figure 9 Models and Transformations in MDA**

For the model transformations, MDA proposes the Query/View/Transformation (QVT) specification which offers a declarative language with both textual and graphical representations. A QVT transformation consists of one or more relations to relate the source and target model elements by declaring the constraints that must be satisfied by the elements. When the transformation is executed, these relations are verified and enforced by manipulating the target model.

MDA approach provides a new perspective in software development by using the model transformations to move from abstract descriptions of some aspects of a system to more detailed and concrete models, and eventually to the code. It classifies different kinds of models allowing the system architects and developers to view the system from different perspectives. By separating the PSMs from the domain and analysis models, it also aims to increase the long-term productivity of high-level PIMs and CIMs  by keeping them away from the refinements on the platforms [49].

## 3.2. ATLAS Model Management Architecture

Another successful MDE implementation is the ATLAS Model Management Architecture (AMMA) which presents a complete set of tools and technologies to support the modeling process in software development [50]. It defines a lightweight architecture similar to a software factory as described in [51]. In order to introduce models to the AMMA, a Kernel Meta-meta-model (KM3) is specified for describing meta-models. KM3 mainly aims to specify the domain-specific languages (DSL) which are designed to be used for a particular set of tasks in a domain, in contrast to general-purpose languages used for multiple application domains[46]. A graphical representation of KM3 constructs is illustrated in Figure 10. Each meta-model based on KM3 is specified as a *Metamodel* instance, which may include one or more *Packages*. Each package, then, includes a number of *ModelElements* to be defined in the modeling process. An example meta-model in KM3 textual format for specifying XML documents is shown in Figure 11.

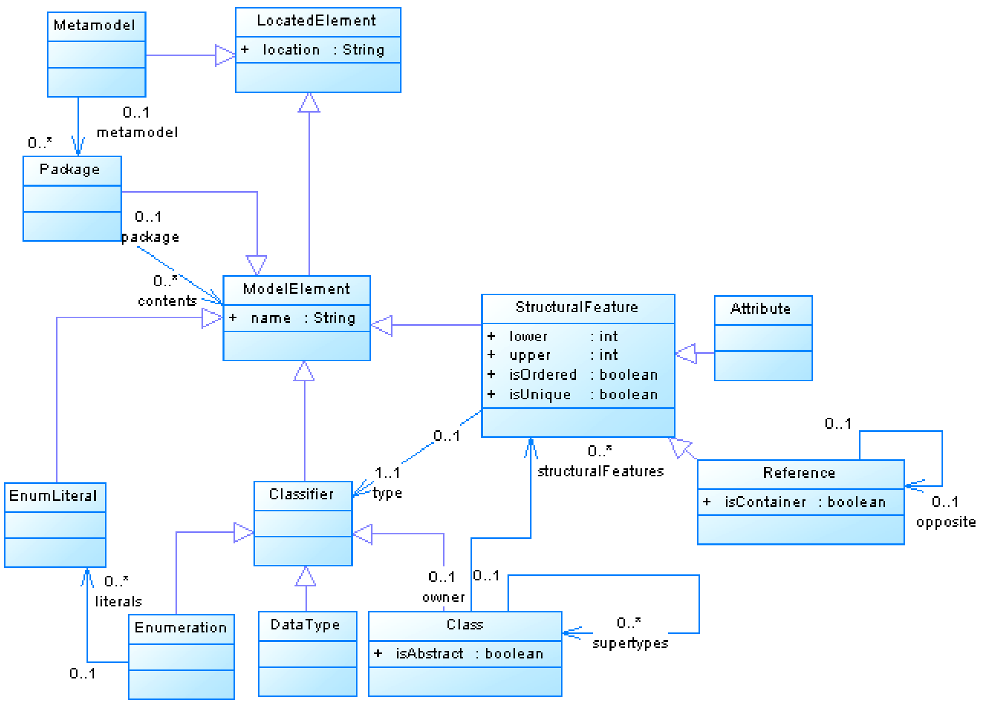


**Figure 10 Graphical Representation of KM3 Meta-meta Model**

```
package XML {
  abstract class Node {
      attribute startLine[0-1] : Integer;
      attribute startColumn[0-1] : Integer;
      attribute endLine[0-1] : Integer;
      attribute endColumn[0-1] : Integer;
      attribute name : String;
      attribute value : String;
      reference parent[0-1] : Element oppositeOf children;
  }

  class Attribute extends Node {}

  class Text extends Node {}

  class Element extends Node {
      reference children[*] ordered container :
        Node oppositeOf parent;
  }

  class Root extends Element {}
}

package PrimitiveTypes {
  datatype Boolean;
  datatype Integer;
  datatype String;
}
```

**Figure 11 An Example Meta-model in KM3 for specifying XML Documents**

Once the models are specified through their meta-models in KM3 format, AMMA utilizes the model transformations in ATLAS Model Transformation Language (ATL). The abstract syntax of ATL is defined by its own meta-model which presents a hybrid language including declarative and imperative constructs to specify unidirectional transformations [52].

A transformation starts with the *module* specification indicating the source and target models. Then, a number of *helpers* can optionally be specified in order to perform navigation over source models or associate read-only named values to source model elements. The main components in an ATL specification are the transformation *rules* which are used to express the actual transformation logic. A *rule* states the source and target patterns to be matched in the models. During the execution of the transformation, the source pattern is evaluated to a set of matches in source models and the corresponding target pattern is created in the target model. Figure 12 illustrates an example transformation from a class model to a relational database.

```
module Class2Relational;
create OUT : Relational from IN : Class;

helper context String def: firstToLower() : String =
  self.substring(1, 1).toLower() +
  self.substring(2, self.size());

helper def: objectIdType : Relational!Type =
  Class!DataType.allInstances()
  ->select(e | e.name = 'Integer')->first();

rule Class2Table {
  from
    c : Class!Class
  to
    out : Relational!Table (
      name <- c.name,
      col <- Sequence {key}
        ->union(c.attr->select(e | not e.multiValued)),
      key <- Set {key}
    ),
    key : Relational!Column (
      name <- 'objectId',
      type <- thisModule.objectIdType
    )
}

rule DataType2Type {
  from
    dt : Class!DataType
  to
    out : Relational!Type (   name <- dt.name )
}
…
```

**Figure 12 An example ATL Transformation**

# 4. SEMANTIC WEB TECHNOLOGIES

The Semantic Web [53] technologies allow the information to be represented and exchanged through formal techniques facilitating the processing of the descriptions on the Web. Semantic Web adopts the idea of ontology, which is previously used in Artificial Intelligence and Database communities, in order to formally model a conceptualization and enable knowledge sharing between information resources [54].

The recent demand in Semantic Web ontologies has increased as a result of the growing need for knowledge management on a global scale. The studies to provide a standard ontology language for Web have been pioneered by the World-Wide Web Consortium (W3C). Based on the existing Web standards such as Extensible Markup Language (XML), Unicode and Uniform Resource Identifier (URI), various Semantic Web languages for ontology specification (e.g. RDF(S), OWL), query (e.g. SPARQL, RDQL), and rules languages (e.g. SWRL, F-Logic) are specified. In this section, an overview of these languages is provided as the foundations of the Semantic Web. Later, the logic languages that form the backbone of the Semantic Web languages are introduced. Additionally, existing tools and platforms are presented in order to guide the semantic-based software development.

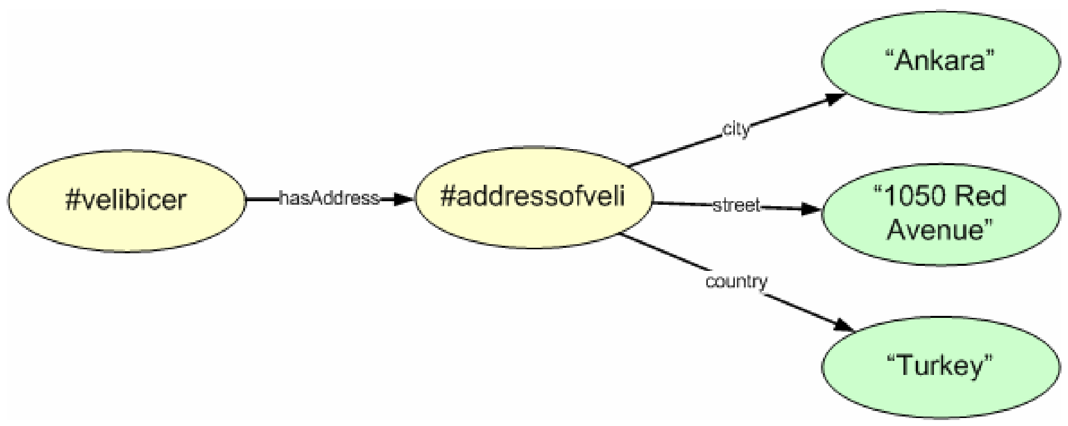
## 4.1. Resource Description Framework (RDF)

The Resource Description Framework (RDF) [55] is the first language developed specifically for the Semantic Web. It uses XML for syntactical representation and URI for resource identification. As the name implies, RDF aims to add a machine-processable metadata to the resources on the Web.

RDF describes the resources by the RDF-statements, which are actually the subject–predicate–object triples. Subject identifies the thing the statement is about.

The property or characteristics of the subject is called the predicate which relates the subject to a value of that property. This value is, then, called the object.

An object of a triple can, in turn, function as the subject of another triple, yielding a directed labeled graph, where resources (subjects and objects) correspond to nodes, and predicates correspond to edges. An example RDF graph and the corresponding triples are shown in Figure 13.



```
#velibicer          hasAddress  #addressofveli
#addressofveli      city        "Ankara"
#addressofveli      street      "1050 Red Avenue"
#addressofveli      country     "Turkey"
```

**Figure 13 An example RDF graph and corresponding triples**

## 4.2. RDF Schema (RDFS)

RDF Schema (RDFS) provides the basic axioms and concepts to define a lightweight ontology to describe RDF vocabularies [56]. Actually, it extends the RDF with the expressions to define classes, class hierarchies, properties, property hierarchies and some restrictions. Each RDF document, therefore, can be interpreted according to the RDFS ontology it conforms. An example is-a hierarchy is depicted in Figure 14.

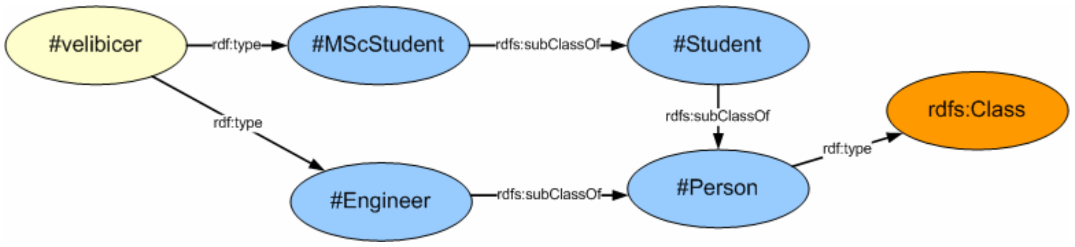


**Figure 14 An example RDFS ontology**

RDFS depends on the RDF and XML as a representation mechanism. Although the definitions of basic concepts such as classes, properties and is-a hierarchies can be easily defined in RDFS, it is not very expressive compared with many other ontology languages. This is the main motivation for developing more expressive languages (e.g. Web Ontology Language) based on RDFS principles.

## 4.3. Web Ontology Language (OWL)

The Web Ontology Language (OWL) [57] extends the RDFS and RDF languages in order to provide an expressive ontology language for Semantic Web. Unlike the RDF(S) triples, it provides additional constructs and vocabularies as axioms or assertions. OWL describes the structure of a domain in terms of classes and properties. Classes can be names (URIs) or expressions. Furthermore, the following set of constructors is also provided for building more complex class expressions:

- *owl:intersectionOf* is used to link a class to a list of class descriptions as their intersection. In other words, the intersection class represents the individuals that are also the instances of all class descriptions in the list.
- *owl:unionOf* is used to link a class to a list of class descriptions as their union.The union class represents the individuals that are also the instances of at least one of classes in the list.
- *owl:complementOf* is used to state a class that represent exactly those individuals that do not belong to the class that is the object of the statement.

In OWL, properties can have multiple domains and multiple ranges. Multiple domain (range) expressions restrict the domain (range) of a property to the intersection of the class expressions. Two types of properties exists according to the range: Object property links a class to another whereas data type property links a class to a data value.

Additional axioms are used to make it possible to assert subsumption or equivalence with respect to classes or properties. The following are the some of axioms used in OWL: rdfs:subClassOf, owl:equivalentClass, rdfs:subPropertyOf, owl:equivalentProperty, owl:disjointWith, owl:sameAs, owl:differentFrom, owl:inverseOf, owl:transitiveProperty, owl:functionalProperty, owl:inverseFunctionalProperty.

OWL has three sublanguages with different power of expressiveness- OWL Lite, OWL DL, and OWL Full:

- *OWL Lite* is the least expressive sublanguage of OWL which is mainly used for the classification and simple constraint specification. It supports basic cardinality restrictions, local range restrictions, existential restrictions, equality, and various types of properties (inverse, transitive, and symmetric).
- *OWL DL* adds full support for (classical) negation, disjunction, cardinality restrictions, enumerations, and value restrictions compared to OWL Lite. The element “DL” comes from the resemblance to an expressive description logic language.
- *OWL Full* allows both the specification of classes-as-instances and the use of language constructs in the language itself, which thereby modifies the language.

OWL uses RDF/XML as its normative syntax. An example ontology is shown in Figure 15.

```
<rdf:RDF>
    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:ID="Engineer">
        <rdfs:subClassOf rdf:resource="#Person"/>
    </owl:Class>
    <owl:Class rdf:ID="MScStudent">
        <rdfs:subClassOf rdf:resource="#Student"/>
    </owl:Class>
    <owl:Class rdf:ID="Person"/>
    <owl:Class rdf:ID="Student"/>
    <MScStudent rdf:ID="velibicer"/>
    <Engineer rdf:ID="velibicer2">
        <owl:sameAs rdf:resource="#velibicer"/>
    </Engineer>
</rdf:RDF>
```

**Figure 15 An example OWL ontology in RDF/XML Syntax**

## 4.4. Description Logics

The current Semantic Web ontologies mainly depend on a number of formal specification techniques that have been studied over the years by using the logical languages for knowledge representation. The use of logic languages provides various benefits in terms of capturing and processing the information. First, this provides expressiveness and machine-processability to enable the derivation of the implicit knowledge from the existing one. Additionally, logical languages enable the specification of the unambiguous statements and allow the application of the formal rules defined in the language during the derivation of implicit information. More importantly, the previous research achieved in the areas of Databases and Artificial Intelligence is reused to form a logical basis for Semantic Web ontologies. Although a number of logical languages, including First-Order Logic (FOL) are proposed as a basis for Semantic Web ontology specifications, the most notable one is the Description Logics (DL) which is a family of languages representing strict subsets of FOL [58].

DL [59] mainly revolves around concepts, roles, and role restrictions. Since it is actually based on FOL, concepts can be seen as unary predicates, whereas roles can be seen as binary predicates. A knowledge base in the basic DL has two parts: the TBox and the ABox. TBox introduces the terminology, i.e., the vocabulary of an application domain, while the ABox contains assertions about named individuals in terms of this vocabulary.

Elementary descriptions are atomic concepts and atomic roles. Complex descriptions can be built from them inductively with concept constructors. In abstract notation, we use the letter A for atomic concepts, the letter R for atomic roles, and the letters C and D for concept descriptions. In basic DL, concept descriptions are formed according to the following syntax rule:

$$\begin{array}{lll} C, D \rightarrow & A \mid & \text{(atomic concept)} \\ & T \mid & \text{(universal concept)} \\ & \bot \mid & \text{(bottom concept)} \\ & C \cap D \mid & \text{(intersection)} \\ & C \cup D \mid & \text{(union)} \\ & \neg C \mid & \text{(negation)} \\ & \forall R.C \mid & \text{(existential restriction)} \\ & \exists R.C \mid & \text{(universal restriction)} \end{array} \qquad (1)$$

Additionally, there are concept axioms which make statements about how concepts are related to each other: $C \subseteq D$ (concept inclusion) indicates that D is more general than C and $C \equiv D$ (concept equivalence) can be interpreted as $C \subseteq D$ and $D \subseteq C$. TBox of knowledge base, actually, involves a number of these axioms to specify a vocabulary. An example TBox is given in Figure 16.

```
Person ≡ ∀hasChild.Person ⊓ ∃hasFather.Father⊓ ∃hasMother.Mother
Person ≡ Man ⊔ Woman
Parent ≡ ∃ hasChild.T
Mother ≡ Woman ⊓ Parent
Father ≡ Man ⊓ Parent
```

**Figure 16 An example TBox**

The individuals in a DL knowledge base are specified as assertions in the ABox. They are of the form i ∈ C or <i,j>∈R, where i,j are individuals, C is an concept and R is a role. An example ABox for the TBox specified in Figure 16 can be written as shown in Figure 17.

```
john ∈ Person
<john, susan> ∈ hasChild
susan ∈ Woman
```

**Figure 17 An example ABox**

## 4.5. Sesame

Sesame [60] provides an architecture for the storage and retrieval of RDF data. As shown in Figure 18, the Sesame architecture is built on top of a number of storage mechanisms such as relational databases, memory storage or native files for the persistent storage. In order to enable this independency from storage devices, a layer, called Storage And Inference Layer (SAIL) is added to the architecture. Actually, SAIL is an interface offering RDF specific methods to the upper layers and handling the conversion from RDF's triple-based representation mechanism to the specific storage devices.

The functional modules of the Sesame uses the SAIL interface in order to support further functionalities to the client applications. Currently, there exist four functional modules which are SeRQL query engine, RQL query engine, admin module and RDF export module.

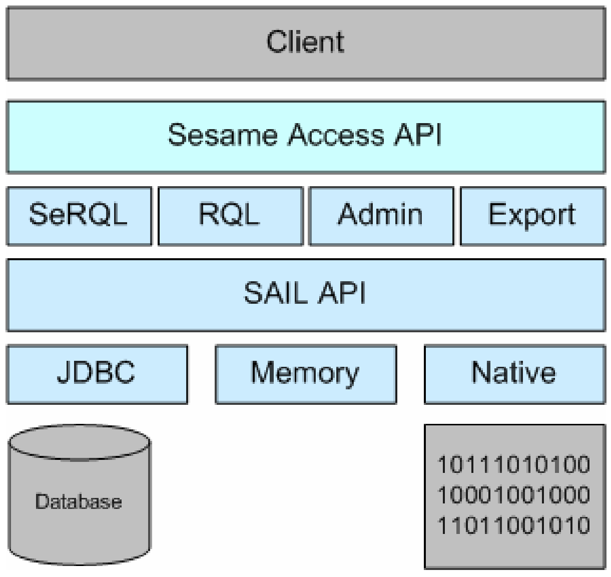


**Figure 18 Sesame Architecture**

Among the query languages supported by Sesame, SeRQL is the most powerful and expressive one which combines the strongest features of the existing query languages. The SQL-like syntax of SeRQL allows us to specify two types of queries: Select and Construct. Select queries return tables of values, or sets of variable-value bindings whereas construct queries return RDF graphs as a set of triples. The values are returned according to the parameters specified with the six clauses of the query, which are SELECT, FROM, WHERE, LIMIT, OFFSET and USING NAMESPACE. Construct queries also use these clauses, but replace the SELECT clause with CONSTRUCT. The first clause in the queries (i.e. SELECT or CONSTRUCT) determines which values should be returned and in what order. FROM clause is used to specify the paths in RDF so that the values are filtered according to the path expression. The third clause in a query is the WHERE clause that is used to specify Boolean constraints on variables. LIMIT and OFFSET enables the retrieval of smaller portions of the results that are generated by the query. Finally, USING NAMESPACE is used to define short prefixes for namespaces, which can then be used in the parameters specified with other clauses. An example select query is specified in Figure 19 in order to select the city of the M.Sc students based on the path expressions defined in Figure 13 and Figure 14.

```
SELECT  Person, City
FROM  {Person} hasAddress {Address};
      {Address} city {City};
      {Person} rdf:type {MScStudent}
```

**Figure 19 An Example Select Query**

Sesame Access API provides access to the Sesame functionalities through HTTP, SOAP, and RMI. This interface can be used either by a client program (e.g. Java application) or another Sesame server for enabling the federation with multiple servers.

## 4.6. Protégé

The Protégé is a well-known, latest tool for ontology development and knowledge management that has been evolving for over the last decade. It provides a graphical and interactive ontology-design and knowledge-base development environment helping knowledge engineers and domain experts to perform knowledge-management tasks. Currently, it supports the development of ontologies in various languages such as RDFS, OWL, Protégé Ontology, rule languages such as SWRL. In addition to highly usable interface and features, two other important features distinguish Protégé from most ontology-editing environments: its scalability and extensibility. Developers have successfully employed Protégé to build and use ontologies consisting of 150,000 frames. Furthermore, the Protégé architecture is constructed in an open, modular fashion. Its component-based architecture enables system builders to add new functionality by creating appropriate plugins. The Protégé Plugin Library contains contributions from developers all over the world. A screenshot of the Protégé is shown in Figure 20.

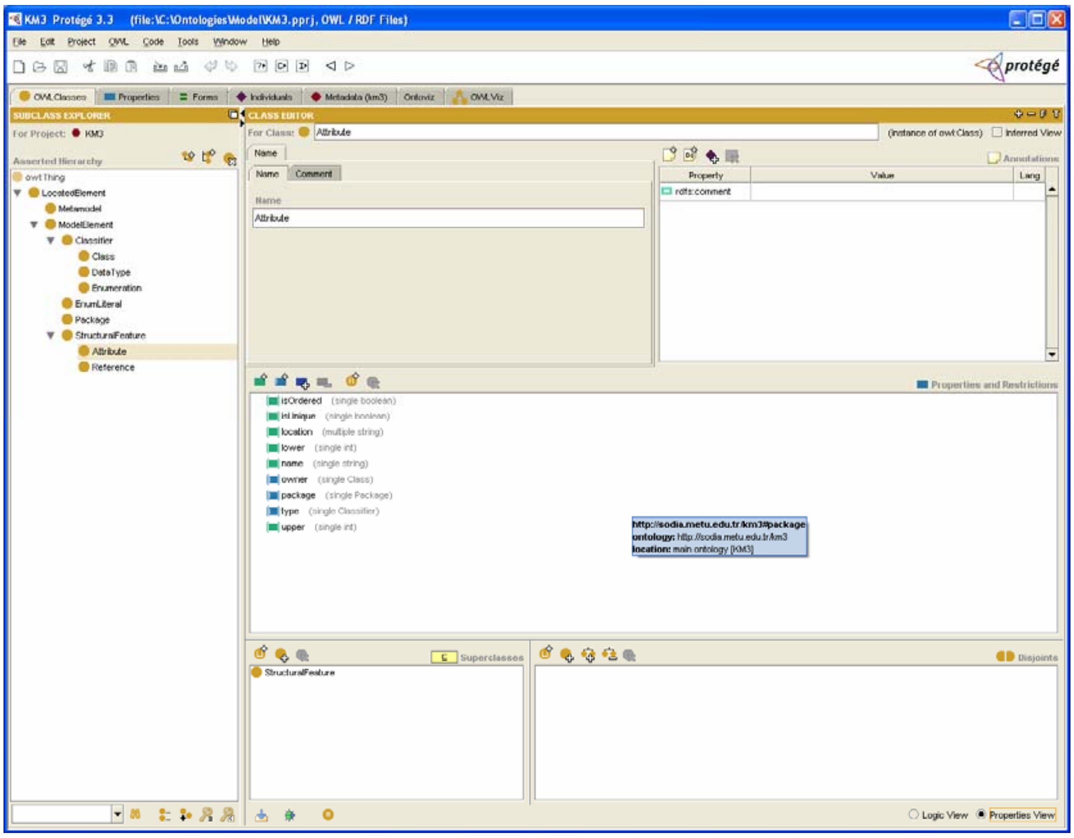


Figure 20 Protégé Ontology Editor

## 4.7. Ontology Mapping with OWLmt

The use of ontology languages such as OWL for representing the information in a machine-understandable way is not enough to achieve interoperability. In an open environment such as Web, it would not be so practical to use very few ontologies shared by many parties. Therefore, there will be many heterogeneous ontologies with overlapping content. Because of this decentralized nature of the Web, each community should have the freedom to use its own ontology definitions to represent the information. This will lead to many ontologies - possibly one for every party- which requires mediation among them by using some new mechanisms. In order to implement this mediation, there is a need for a tool which can specify the ontology mappings between these ontologies. This semantic mapping is an inevitable operation to establish interoperability between agents or services using different ontologies.

OWLmt[61] is developed as a generic tool to specify the mappings between OWL ontologies. Ontology mapping is the process whereby two ontologies are semantically related at conceptual level, and the source ontology instances are transformed into the target ontology entities according to those semantic relations. In this process, we specify the semantic matching between the source ontology and the target ontology which share an overlapping content. It includes the matching between the entities which are concepts, relations and properties such as classes, object properties, or data type properties in OWL. In addition, the ontology can have instances which are defined according to the corresponding ontology to include the actual data in the process. We refer to this collection of instances as Instance Base.

The mapping process is based on the possible operations that can be defined between the source ontology and the target ontology. These operations can include functionalities such as relating two or more classes to a target class, transforming the data values to the corresponding values in target ontology or constructing new relations between the instances in the target ontology according to a specified path in the source ontology. To define this set of functionalities, we create a *mapping schema* which specifies the features of the mapping process such as the definitions of the possible operations, representations of the entities and the relationships among them. The *mapping schema* is also defined in OWL since it should be machine-processable and able to represent the domain of the mapping process. The mapping definitions, conforming to *mapping schema,* are defined through the OWLmt GUI and executed by the mapping engine to transform source ontology instance base to the target ontology instance base. This process is illustrated in Figure 21. The main screen of the tool is also shown in Figure 22.

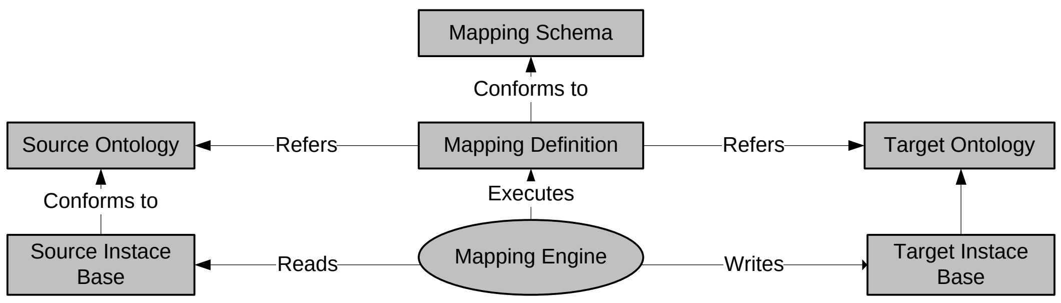


**Figure 21 Ontology Mapping Process in OWLmt**

The following capabilities are provided in OWLmt mapping tool:

- *Matching the source ontology classes to the target ontology classes:* We have developed the following four conceptual mapping patterns to represent the matching between the classes of the source and target ontology classes: *EquivalentTo, SimilarTo, IntersectionOf, and UnionOf.* The identical classes are mapped through *EquivalentTo* pattern. *SimilarTo* implies that the involved classes have overlapping content. How similar classes are further related is described through property mapping patterns. The *IntersectionOf* pattern creates the corresponding instances of the target class as the intersection of the declared source class instances. Similarly, the *UnionOf* pattern implies the union of the source classes' instances to create the corresponding instances of the target class.
- *Matching the source ontology object properties to target ontology object properties: ObjectPropertyTransform* pattern is used to define the matching from one or more object properties in the source ontology to one or more object properties in the target ontology.
- *Matching source ontology data properties to target ontology data properties:* Through the *DatatypePropertyTransform* pattern, the data type properties of an instance in the source ontology are mapped to corresponding target ontology instance data type properties. OWLmt supports a set of basic XPath [62] functions and operators such as concat, split, and substring. In some cases, there is a further need for a programmatic approach in order to specify complex functions (e.g., need to

use if-then-else, switch-case, or for-next). Therefore, we have introduced JavaScript support to OWLmt. By specifying the JavaScript to be used in the *DatatypePropertyTransform* pattern, the complex functions (enriched by the Java SDK libraries) can be applied in the value transformations.

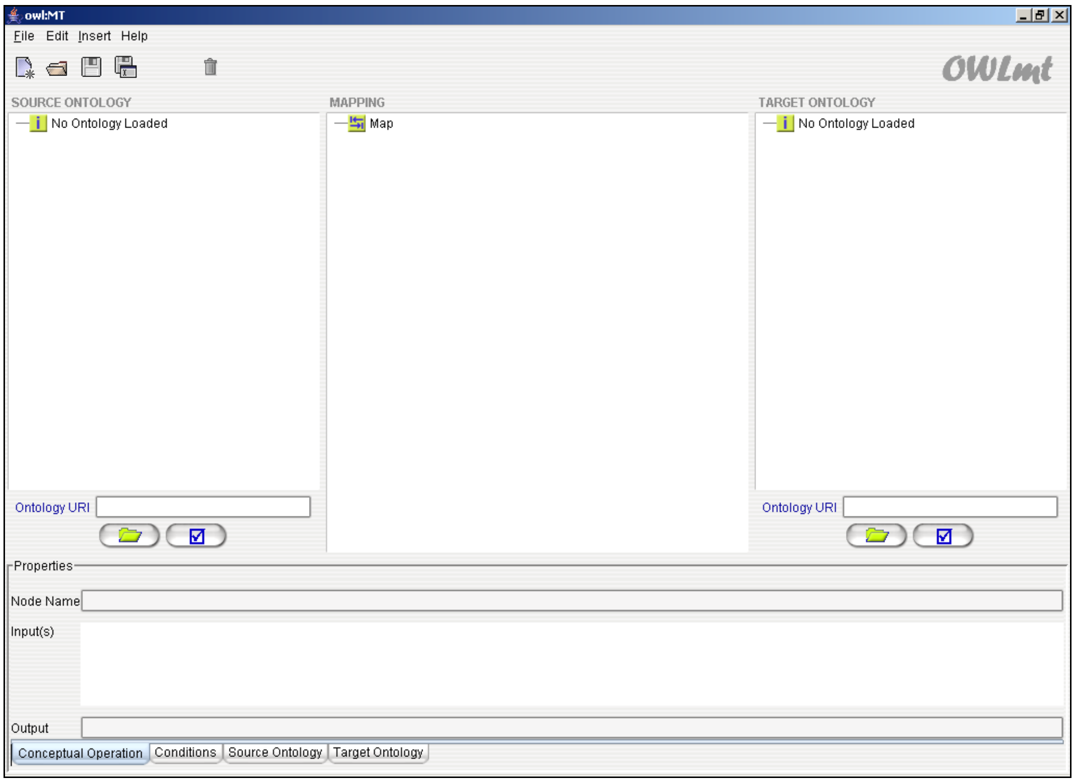


**Figure 22 OWLmt GUI**

# 5. SEMANTIC-BASED MODELLING OF SERVICE-ORIENTED ARCHITECTURES

In this chapter, a process for service-oriented architecture design is presented. So far we have identified the existing technologies and approaches in related fields. In order to present the approach, we start by providing an overview of the complete methodology proposed in this thesis for embedding the modeling approach and introduce the possible steps to be taken to create architectural descriptions. Specifically, the basic steps of the architecture specification process are presented by utilizing many technologies and concepts in a combination.

## 5.1. Overview of the Methodology

We suggest lifting various service-oriented concepts to be represented as ontology-based models in the design process and create an architecture-driven model-based development process by utilizing semantic Web technologies. A range of layers considered in this methodology is depicted in Figure 23. Currently, legacy systems and service-components can be exposed as services as the initial step to construct service-oriented systems. These services are complemented with the upper layer service-oriented technologies like composition to create more operational and business oriented functions. As introduced in previous chapters, the crosscutting layers like management, security and the related middleware technologies are also major parts of the architecture.

Software engineering layers to be detailed in following sections extend this picture by enabling a design from a system engineering perspective. Software architecture layer on the top of Figure 23 enables an abstract view of the system in which the associated service-oriented technologies can be grouped into the corresponding architectural viewpoints. The MDE layer, on the other hand, allows us to represent any service-oriented model and draw connections between the domain-specific abstractions and service-oriented platform. Furthermore, we employ a range of semantic Web techniques and tools such as OWL, Sesame Server, or OWLmt in order to facilitate the description of SOA through ontologies.

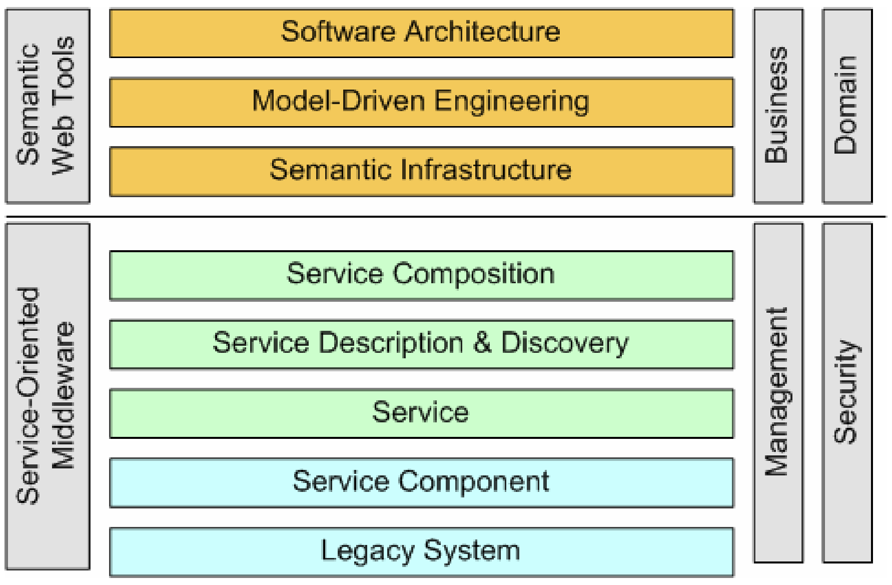


**Figure 23 Service-Oriented Development Hierarchy**

Within the scope of this idea, the service-oriented technologies and platforms can be viewed as a valuable set of reusable assets to be used in realizing distributed enterprise software architectures. To facilitate this, a two-fold methodology is employed as depicted in Figure 24: Platform engineering, which is also the main topic of this thesis, is used to define the features of the platform to construct, collect, and organize existing domain experience and to create a service-oriented architecture specification for building enterprise systems in the form of reusable service-oriented assets [63]. This is a required step, prior to application development, in order to align the existing assets in a particular architecture, produce a service-oriented platform and use the service-oriented technologies effectively during application design. In addition, application engineering is utilized to use the service-oriented platform created as a result of platform engineering to create service-oriented systems based on the business requirements.

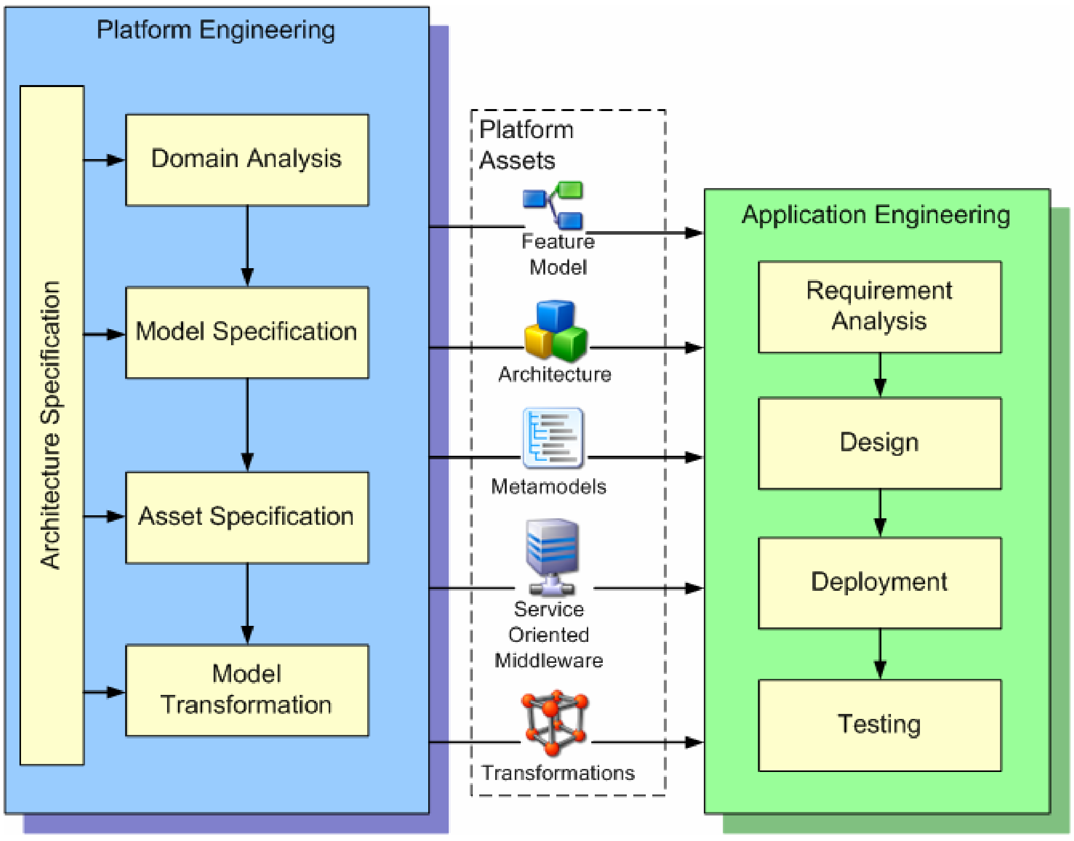


**Figure 24 Overview of Methodology**

The platform and application engineering processes are mostly knowledge-driven activities where various artifacts such as the requirements, architecture, service-oriented design models, platform specifications, and model transformations are needed to be defined in a precise and unambiguous way. The use of ontologies play a key role in this development in terms of ontology-driven development and ontology-enabled architectures [64]. An ontology-based system engineering process offers the following benefits in various stages of the service-oriented system engineering:

- **Meta-programming:** The latest trend in software engineering research on product lines, model-driven engineering, and generative programming is to utilize the automation of software development with the aim of regarding programming as a computation. This requires a meta-application level process, also named as meta-programming, that uses the metadata of the applications to construct, and operate on application level artifacts. Today, Semantic Web ontologies are the most well-known way of capturing and managing this meta-data in terms of concepts and the relationships among them.

- **Mega-modeling:** In addition to models and meta-models of MDE to represent the system design such as business process or data models, the service-oriented tools, platforms, registries, policies and other entities should also be considered in the system development lifecycle. Considering all these entities as resources, Mega-modeling approach aims to enable resource management by representing these global entities as models [50]. Ontologies provide very generic means to represent any entity that is supposed to exist in software development process.
- **Traceability:** Traceability helps stakeholders and system developers understand the many associations and dependencies that exist among various software artifacts created or reused during a software development process. Providing various ontologies to represent these artifacts provides a new way of establishing and using traceability information.
- **Tool Support:** Since the introduction of Semantic Web vision, a number of tools such as parsers, ontology repositories, reasoners and ontology mapping tools are provided in order to support semantic-based application frameworks. By employing a semantic-based infrastructure for software development, enterprise will be able to benefit from these tools in standard and consistent way.
- **Access to the information in a global scale:** The representation of models and related elements through standardized ontologies will enable the enterprises to store the domain knowledge and models in a sharable and processable way. This will enable the enterprises to share these ontologies through special networks (e.g. ecosystems over P2P networks) enabling a model-level reusability.

## 5.2. Architecture Specification

The activity of creating a service-oriented architecture requires the organization of essential concepts and principles in a precise way with the aim of building an architectural description. Although this process shows similarities with the studies achieved in software architecture field since 1990s [65], the development of the emerging approaches and technologies both in service-oriented computing and software engineering requires an expansion in the architecture methodology. Essentially, creating architecture is a transformation from the problem domain to the solution domain [44] where the service-oriented technologies are a part of the solution domain as a set of reusable assets. According to our proposed methodology, an architecture description is created by considering the stakeholders' concerns, problem domain, existing domain knowledge and the solution domain (including SOA assets). This description mainly drives the system development process in the application engineering. This is depicted in Figure 25.

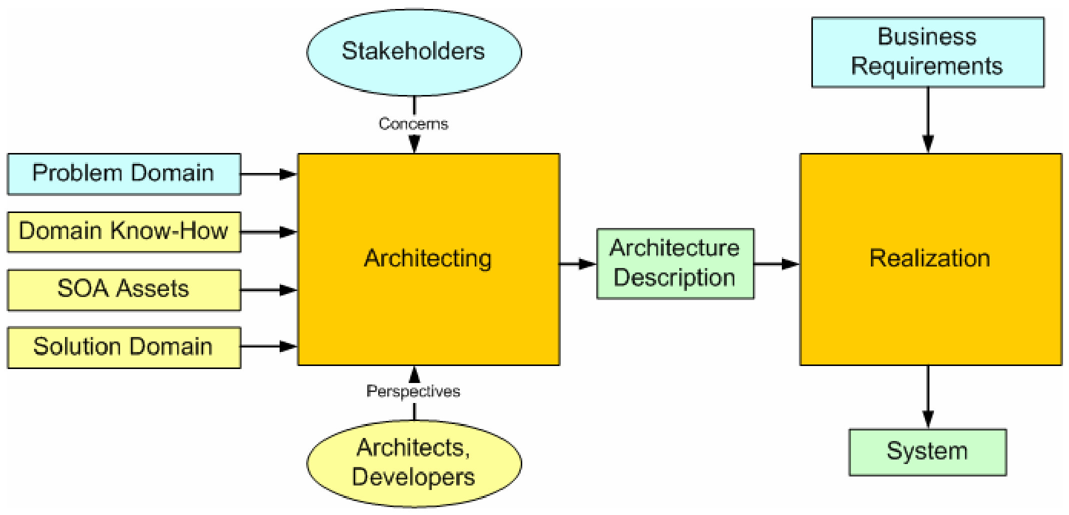


**Figure 25 Architecture Specification Process**

In order to better utilize the architecture specification process, an architecture ontology is developed as illustrated in Figure 26. The benefits of this ontological approach are threefold: First, it offers a formal way to place any service-oriented and software engineering concept in the architecture by specifying the relationships among them. This allows us to use various concepts and approaches ranging from domain analysis to model specification which are clearly linked with each other in a well-defined way. Additionally, it is an initial requirement in automating the system development, since the ontology enables the machine-processability supported by the standard-based tools and platforms. Finally, the existence of a representation mechanism is a necessity for better documentation in the software architecture, otherwise, the design decisions, which are implicitly present, can be quickly lost [22].

We extend the IEEE Std 1471-2000 [21] specification by referring to the ontologies specifically designed for the sub-processes of the platform engineering such as feature models, MDE meta-models or ontology mapping definitions. In fact, the architecture ontology is a higher level abstraction level which stands on top of these ontologies allowing the system architect to unify the efforts from different perspectives. In addition, the logical layers in service-oriented architecture stack are represented as a specific viewpoint in the architecture ontology. For brevity, the upper classes of the IEEE Std 1471-2000 are ignored in Figure 26.

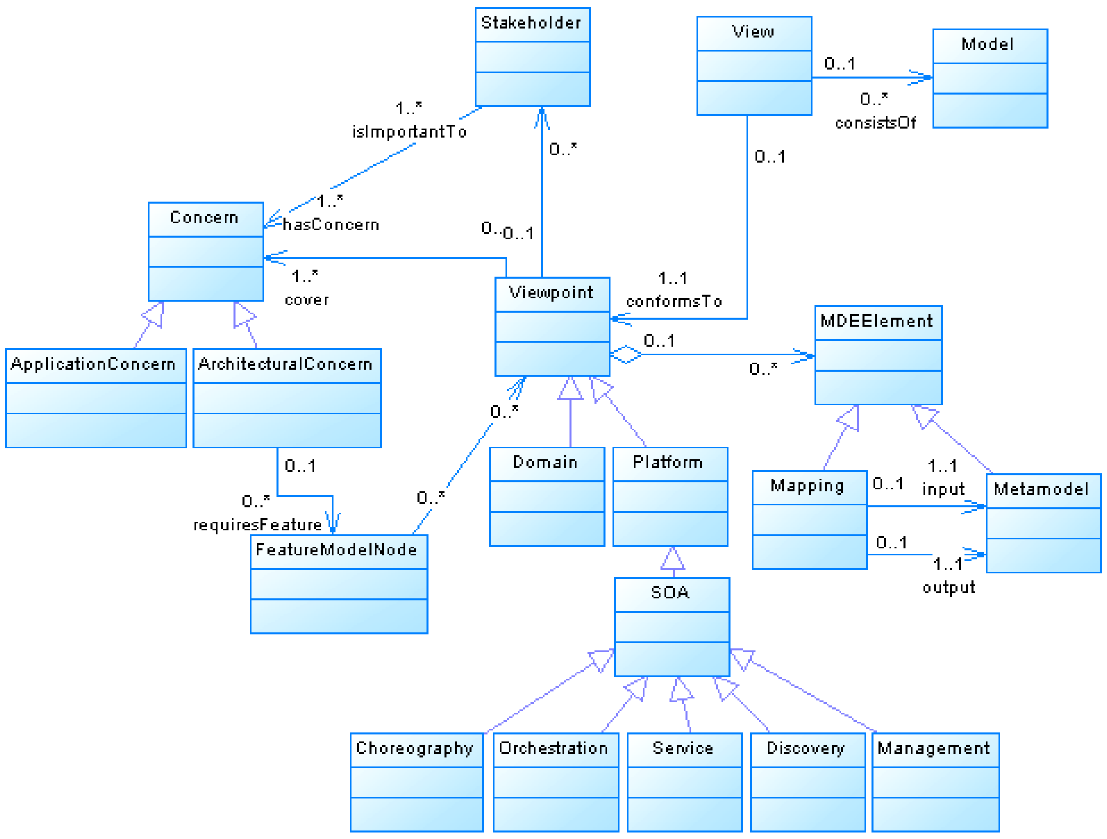


**Figure 26 Graphical Representation of Architecture Specification Ontology**

The platform architecture is driven by the stakeholders' goals and concerns which are represented as instances of the *Concern* class. Two subclasses are also created for representing the platform specific concerns and application specific concerns, namely *ApplicationConcern* and *ArchietecturalConcern*. Each architectural concern is then mapped to one or more features in the feature model to specify distinctive, user-visible aspect, quality or characteristic of the service-oriented platform. Features are created as instances of *FeatureModelNode* class that is included in the feature model ontology presented in the next section. The feature modeling based on the stakeholders' concerns is a good technique to define the scope of the service-oriented platform since it does not intend to solve all the problems of the domain but specifically the ones addressed by the stakeholders. By mapping the features to the corresponding viewpoints, realization of the features is accomplished from an architectural point of view. Figure 27 illustrates these relationships among the concerns, features and viewpoints.

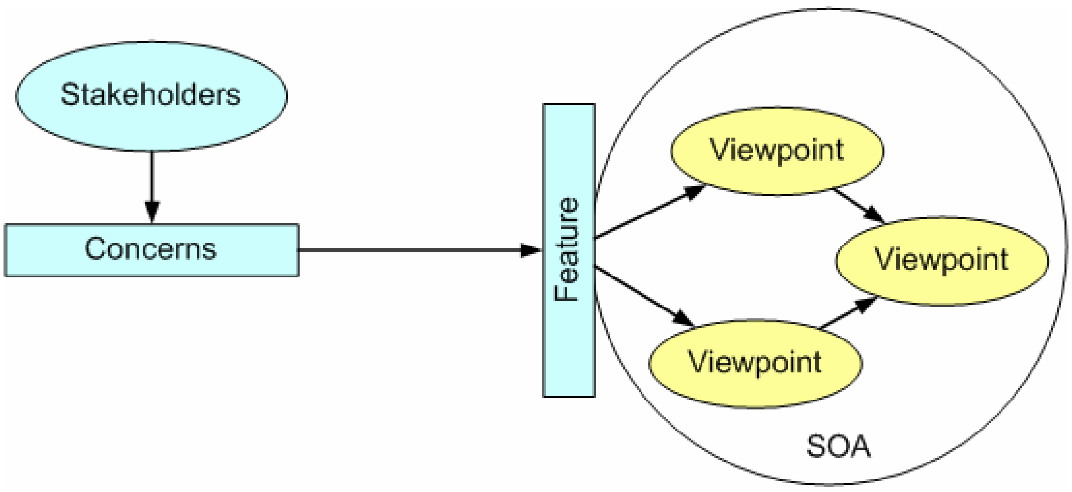


**Figure 27 Relationships among Concerns, Features and Viewpoints**

Viewpoints are used to organize the tools, processes, and assets in the architecture description. Although IEEE Std 1471-2000 provides a standardized way to specify the viewpoints, it does not define a fixed set of viewpoints for any specific methodology. Therefore, we extend this approach by identifying particular viewpoints that may exist in a service-oriented context. First, considering the platform independent and platform specific focus on system aspects, we identify two types of viewpoints, namely *Domain* and *Platform*. Furthermore, five main service-oriented viewpoints are specified as subclasses of the *Platform* viewpoint based on the common logical layers in service-oriented stack and analysis provided in [33] since we consider the service-oriented technologies and specification as a platform to realize service-oriented systems.

As the application of a model-driven approach, a viewpoint is organized as a set of MDE elements such as meta-models and transformations among them. Each model specification is introduced to the platform as semantic-based meta-models based on KM3 meta-meta model of AMMA platform [46]. Transformations among the meta-models are also implemented as ontology mapping since the meta-models and models are captured as ontologies. The details of the modeling approach are presented in the following sections.

## 5.3. Domain Analysis

The first sub-process of the architecture specification is the domain analysis which encompasses the activities for specifying the requirements of the service-oriented platform. These requirements, however, differs from the requirements of a particular application to be realized in the application engineering process. The output of the domain analysis sub-process is very crucial to decide the capabilities of service-oriented platform during the creation of the architecture.

For domain analysis, we use the feature models to represent the service-oriented platform features in a hierarchy. It is a well-known technique which is extensively used for domain analysis to scope and develop software product lines [20, 66] or domain-specific languages [51, 63]. In order to represent the feature models, a semantic-based approach is used as described in [67, 68].

Cardinality-based feature models aims to identify the system properties by extending a number of already existing approaches to the feature modeling[69]. An example feature model is depicted in Figure 28 which shows the capabilities of a clinical mobile application designed for doctors. It consists of a number of nodes and links that bind these nodes to form a hierarchy. The features can be classified according to their position in the model: *Root, solitary, and group*. Similarly, the links can be grouped into two, namely *sub-feature link* and *group link*. A sub-feature link associates a solitary feature to its parent, whereas group link associates a group feature to the features to be included in the link. There are also two types of cardinalities: *Feature cardinality* is used to qualify a solitary feature to specify how often the solitary feature can be copied. *Group cardinality*, on the other hand, is used to indicate the number of the features to be chosen in a group. In feature model diagram, [m..n] is used to indicate feature cardinalities and <m..n> is used to specify group cardinalities, where m and n are integer numbers.

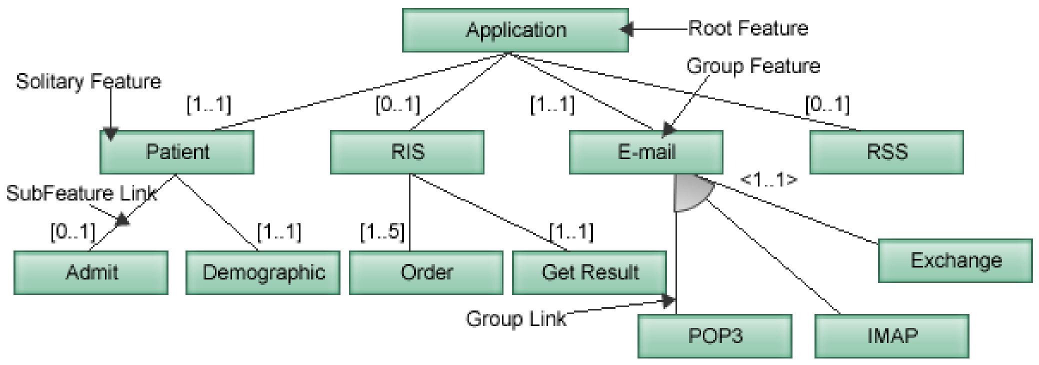


**Figure 28 Cardinality-based Feature Model Example**

In order to capture the system features, we provide a feature model ontology as presented in Figure 29. Every node in a feature model is represented as a sub-type of the *model node* class. There are three types of specialization for the *model node* class in the meta-ontology which are the *root node, feature node*, and *group node* to represent the root features, solitary features and the feature groups as proposed in cardinality-based feature models. In addition to these, any possible extension to the nodes in the feature models can also be included as a subclass of the *model node* class.

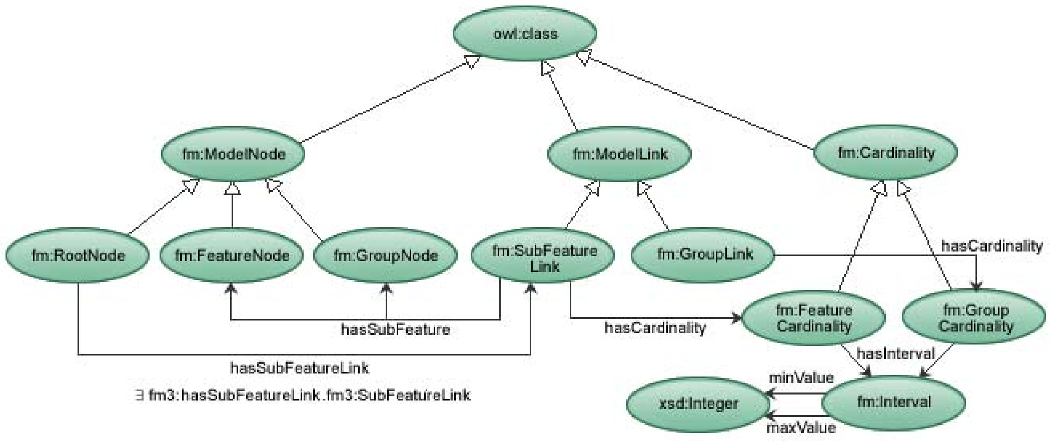


**Figure 29 Overview of Feature Model Ontology**

For the representation of the links in the feature model, an abstract *model link* class is also created. It is the basis for all the relationships among the nodes of the feature model. One of the crucial associations in a feature model is the *sub-feature link* through which the parent-child relation between the features is represented. Therefore, we derive a class from the *model link*, namely *sub-feature link*, in our meta-ontology to denote this connection. The *sub-feature link* class has two important properties: The "*hasSubFeature*" property relates the *sub-feature link* class to a particular solitary feature which can be of type *feature node* or *group node*. Additionally, "*hasCardinality*" property associates the link with the corresponding feature cardinality. This is required due to the fact that cardinality-based feature models requires the solitary features to be specified with the feature cardinalities in order to indicate the number of times a feature can occur in the configuration. The properties are also similar for the *group link* class which is used to relate nodes in a feature group. However, some of its properties are ignored in the figure for the brevity. Observe that "*hasCardinality*" points to another subclass of *Cardinality*, namely *group cardinality*, since it is specified with only one interval whereas the *feature cardinality* may include more than one interval.

Feature cardinality refers to a sequence of intervals in which the minimum and maximum values of the interval is denoted. The *Cardinality* class has a "*hasInterval*" property whose range points to another class called *Interval* to specify the minimum and maximum values of the cardinality.

The feature model ontology enables us to specify the feature models in a formal way to represent the system properties. In fact, architecture ontology refers to this ontology to associate the concerns with the features. Therefore, we can draw equivalence between the feature model node class of architecture ontology and model node of feature model ontology to connect them (arch:FeatureModelNode≡fm:ModelNode).

## 5.4. Model Specification

Model-Driven Engineering offers a new perspective in software development by replacing the previous idea of object composition with model transformation. MDE is an ideal complement to the service-oriented computing in order to better utilize the service-oriented technologies and concepts for enterprise software development. It provides the means to create service-oriented models and rules for the management within system architecture. We develop a semantic-based meta-modeling architecture to specify and design service-oriented systems as a set of DSLs. The DSLs are used within the architectural viewpoints as shown in Figure 26.

In order to specify the DSLs, KM3 meta-meta model [46] of AMMA platform is represented in OWL as depicted in Figure 30. This is a generic ontology introducing the basic concepts and relations in order to specify any meta-model as sub-ontology. The *Metamodel* class encapsulates one or more *Package*s which includes the actual content to be defined in the meta-model in terms of *Classifiers* and *StructuralFeatures*. Therefore, the *Metamodel* class, containing one or more *Packages*, represents a DSL as a modeling asset in an architectural viewpoint.

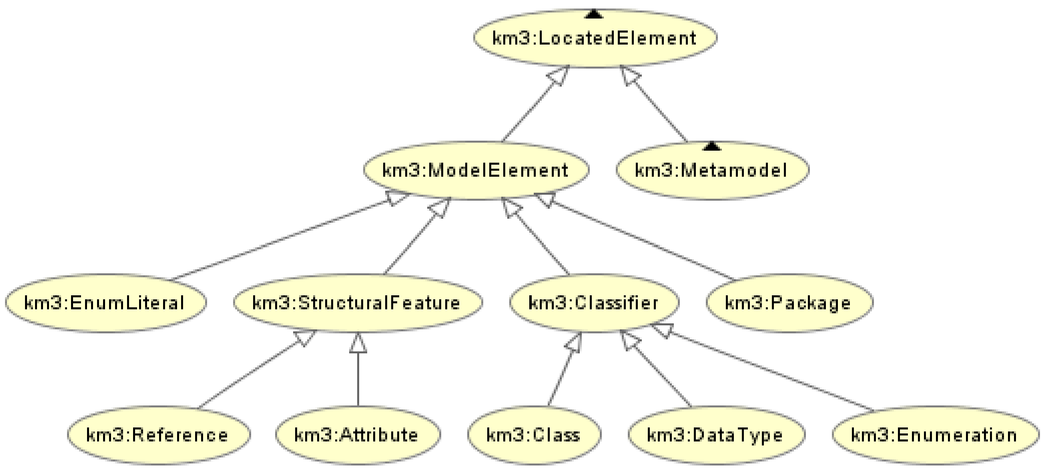


**Figure 30 Kernel Meta-meta-model Class Hierarchy**

Figure 31 shows how an XML DSL meta-model is defined as sub-ontology of KM3. Each meta-model element to be specified in XML DSL is defined as a subclass (rdfs:subClassOf) of its type in KM3. For example, the classes to be included in XML DSL (e.g. Node, Element, Attribute, Text, and Root) are all inherited from the *km3:Class* which specifies their type in the meta-model.

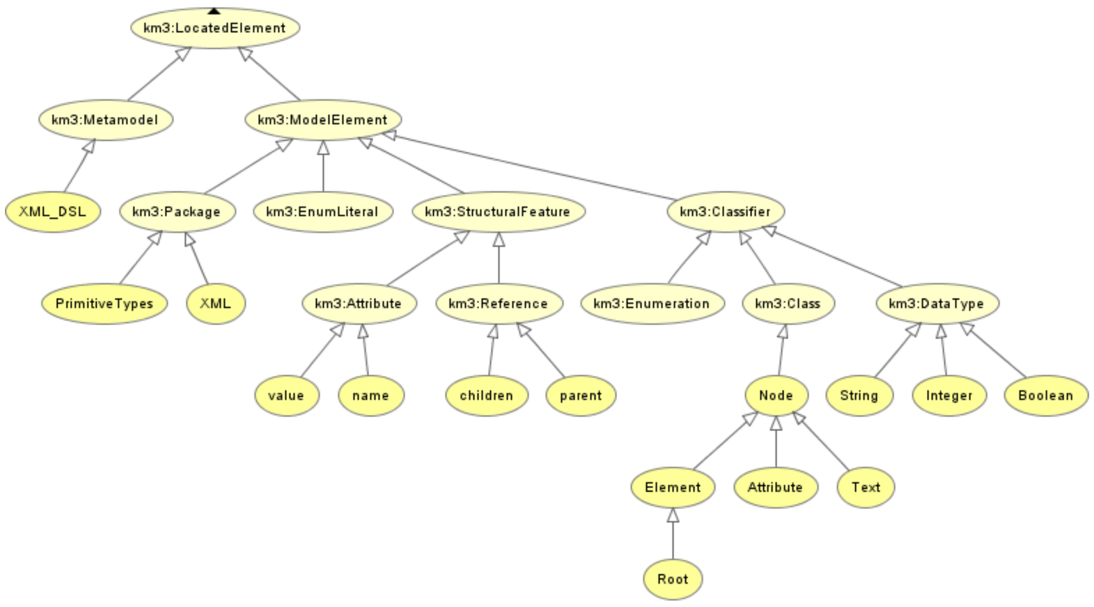
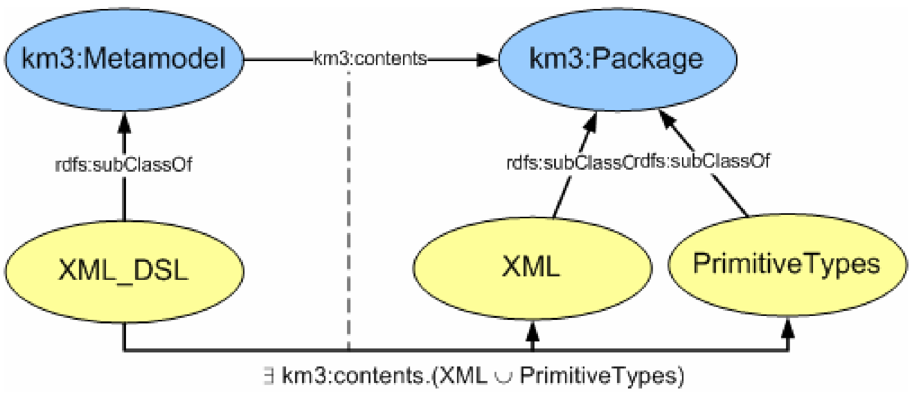


**Figure 31 Example XML Meta-model defined as sub-ontology**

The relationships among the meta-model elements are achieved by applying the OWL restrictions on the inherited properties from KM3 ontology. For example, the *km3:contents* property, which relates a *Metamodel* to its *Package* contents, is restricted for the XML_DSL class to have some of its contents in the types of *XML* and *PrimitiveTypes* at the model level (instance-level). This is stated as a *owl:someValuesFrom* restriction (∃km3:contents.(XML∪PrimitiveTypes)) as shown in Figure 32 with the corresponding RDF triples. Considering this restriction, we can define the XML_DSL meta-model class in DL as:

$$\text{XML\_DSL} \subseteq \exists \text{km3:contents.}(\text{XML} \cup \text{PrimitiveTypes}) \cap \text{km3:Metamodel} \qquad (2)$$



| | | |
|---|---|---|
| #XML_DSL | rdfs:subClassOf | km3:Metamodel |
| #XML | rdfs:subClassOf | km3:Package |
| #PrimitiveTypes | rdfs:subClassOf | km3:Package |
| #XML_DSL | rdfs:subClassOf | #C1 |
| #C1 | owl:onProperty | km3:contents |
| #C1 | owl:someValuesFrom | #XML |
| #C1 | owl:someValuesFrom | #PrimitiveTypes |

**Figure 32 Specifying the connections among meta-model elements**

Similarly, XML class, which is a Package in XML DSL specification, includes a number of Classifiers as basic building elements of the XML documents. The XML Package is defined with the following definitions:

$$\text{XML} \subseteq \exists \text{km3:packageContents.( Node} \cup \text{Element} \cup \text{Attribute} \cup \text{Text} \cup \text{Root)} \qquad (3)$$

$$\text{XML} \subseteq \text{km3:Package} \qquad (4)$$

Additionally, the properties of the meta-model elements are defined as *StructuralFeatures* which can be in two types, namely *Attribute* and *Reference*. These property classes refers a *km3:Class* definition (i.e. owner of the property) to a range which can be another *km3:Class* or a *km3:DataType*. For example, in XML DSL, an *Element* can contain one or more *Nodes* as its children allowing us to create hierarchies in XML documents. For this purpose, we specify a *children* class as a subclass of *Reference* and specify the required restrictions to relate the *Elements* to the *Nodes* as shown in Figure 33. Thus, the *children* reference can be defined as:

$$\text{children} \subseteq \forall \text{ km3:type.Node} \cap \text{km3:Reference} \qquad (5)$$

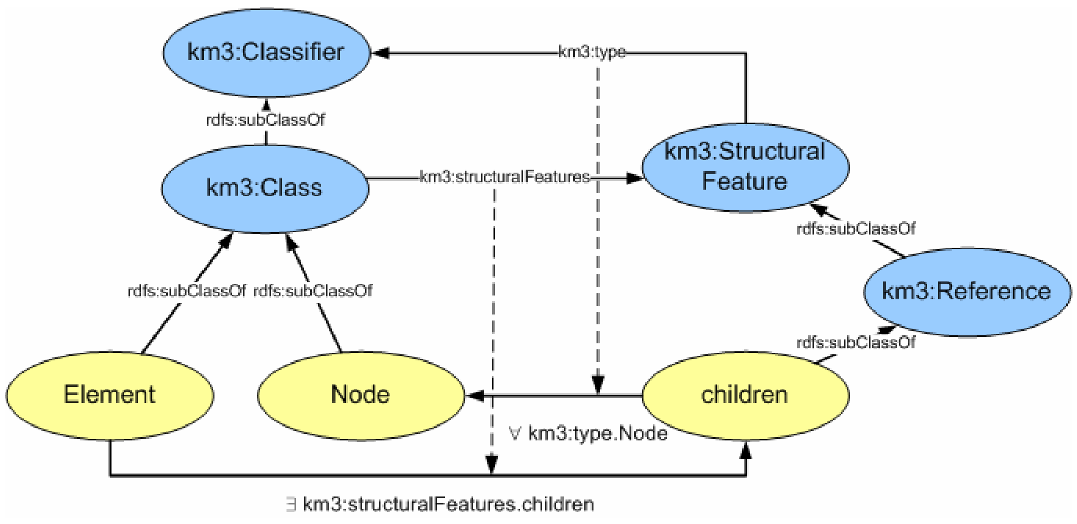


**Figure 33 Example "children" Reference from Element to Node**

Once the meta-models are specified in OWL based on KM3, they can be managed through an ontology repository for storing, querying and managing the modeling metadata. Sesame Server is employed for this purpose to map the model specifications to a persistent repository and provide the functionality for the management of the model data.

As Sesame Server stores the ontologies as RDF triples (subject, predicate, object), the models defined in OWL can be submitted to the repository through the Sesame GUI or API. Thus, Sesame acts as a metadata database and provides distinct query capabilities as described in section 4.5 in order to enable the enterprises to build software development tools and frameworks on top of it.

Before submitting any model to the repository, however, it should be populated with the base KM3 ontology depicted in Figure 30 because any model specification derives from this ontology. We can also query the repository based on this ontology to gather the metadata about the model specifications available. For instance, in order to get a list of the meta-models, a query to retrieve the subclasses of *Metamodel* class of KM3 ontology can be specified as shown in Figure 34. Additionally, to retrieve the *Packages* included in a particular Metamodel (i.e. XML_DSL in this case) can be retrieved with the query given in Figure 35. In these queries, we introduce the namespaces of the metamodel ontologies with the USING NAMESPACE directive assuming that each ontology is described with a unique namespace.

```
SELECT DISTINCT
   Metamodel
FROM
   {Metamodel} rdfs:subClassOf {km3:Metamodel}
USING NAMESPACE
    km3 = <http://sodia.metu.edu.tr/km3#>
```

**Figure 34 SeRQL query to retrieve metamodels from repository**

```
SELECT DISTINCT
   Pck
FROM
 {xml:XML_DSL} rdfs:subClassOf {K}
      owl:onProperty {km3:metamodelcontents},
 {xml:XML_DSL} rdfs:subClassOf {K} owl:someValuesFrom {Pck}
USING NAMESPACE
    km3 = <http://sodia.metu.edu.tr/km3#>,
    xml = <http://sodia.metu.edu.tr/xml#>
```

**Figure 35 SeRQL query to retrieve packages in XML_DSL metamodel**

Similarly, we can obtain the Package concepts or class properties with the queries shown in Figure 36 and Figure 37, respectively.

```
SELECT DISTINCT
   Content
FROM
 {xml:XML} rdfs:subClassOf {K} owl:onProperty {km3:contents},
 {xml:XML} rdfs:subClassOf {K} owl:someValuesFrom {Content}
USING NAMESPACE
    km3 = <http://sodia.metu.edu.tr/km3#>,
    xml = <http://sodia.metu.edu.tr/xml#>
```

**Figure 36 SeRQL query to retrieve contents of XML package**

```
SELECT DISTINCT
   P
FROM
 {xml:Node} rdfs:subClassOf {K}
      owl:onProperty {km3:structuralFeatures},
 {xml:Node} rdfs:subClassOf {K} owl:someValuesFrom {P}
USING NAMESPACE
    km3 = <http://sodia.metu.edu.tr/km3#>,
    xml = <http://sodia.metu.edu.tr/xml#>
```

**Figure 37 SeRQL query to retrieve the properties of Node class**

## 5.5. Using Ontology Mappings for Model Transformations

By representing the meta-models and models through the ontologies, we can employ the semantic Web technologies in MDE. Ontology mapping provides a generic methodology to define the transformations between the models by lifting the transformation idea to the semantic Web techniques. This presents a new area of use for ontology mapping in addition to mapping healthcare data and P2P network messages as presented in [70, 71].

OWLmt tool is used to define the mappings between the meta-models defined according to KM3 ontology. This results in a mapping definition to be executed for transforming any model conforming to the source meta-model into the target model. This is possible since the model transformations, actually, utilizes the idea of pattern matching (i.e. rules in ATL) in order to query the source model and create the corresponding elements in target meta-model. This is similar to the semantic queries in OWLmt created as a result of the class and property relationships. A comparison between ATL and the current version of OWLmt in terms of the model transformation features introduced in [47] is given in Table 1.

**Table 1 Feature Comparison between ATL and OWLmt**

| Feature | ATL | OWLmt |
|---|---|---|
| Domain Language | KM3 | OWL |
| Directionality | Unidirectional | Unidirectional |
| Transformation Specification | Module | Mapping Definition |
| Source-Target Relationship | In, Out, In/Out | In, Out |
| Traceability Links | Dedicated Automatic Support | User Based |
| Rule Application Strategy | Non-deterministic | Interactive |
| Rule Iteration | Recursion | Not supported |
| Rule Selection | Rule Source Guard | Explicit Condition plus Reasoning |
| Rule Scheduling Form | Implicit, Explicit | Implicit |
| Syntax | Textual | Textual, Graphical |
| Code Reuse | Helper Libraries | Javascript Libraries |

OWLmt uses OWL as a domain language. Similar to ATL, it supports unidirectional transformation providing one-way mapping by creating target ontologies based on the source. Although some features are not currently supported, one of the most important benefits for using OWLmt is its semantic-based structure to represent the transformations as ontologies and to complete the architecture presented in this section. A mapping specification is presented in the following chapter.

# 6. SERVICE-ORIENTED SYSTEM ENGINEERING IN PRACTICE

The aim of this research is to provide a semantic-based model driven design for realizing SOAs that better utilizes the service-oriented technologies and concepts throughout the development lifecycle. In this chapter, we will present a case study for the healthcare domain in order to show the use of our method. To begin with, we present an overview of the healthcare informatics in order to show how it can be so heterogeneous due to various standards and systems coexisting in a hospital setting. We also introduce an application requirement for the healthcare professionals to perform various tasks in their daily job. Using the method to design this sample system architecture is presented in the last section.

## 6.1. Healthcare Scenario

Healthcare is one of the few domains in which the software systems play a key role in terms of optimizing the processes, facilitating the information sharing and enabling the coordination and management of healthcare services. Healthcare information systems are evolving in a rapid pace resulting from the paradigm shifts both in healthcare domain and IT technologies. Since the introduction of the Hospital Information Systems (HIS) two decades ago, the change has taken place from the isolated systems that store and retrieve data to the interconnected healthcare infrastructures utilizing cross-enterprise processes and information sharing. Today, the increasing demands of the healthcare domain such as enabling patient mobility, downsizing the hospitals, and improving the patient care raises the needs for institutional and (inter-)national healthcare information system strategies and deploying new architectural styles as addressed in [72]. In other words, a healthcare information system infrastructure, today, can be regarded as an evolving, dynamic entity that is being continually shaped by economic, political, technological, and social forces.

Over the years, various types of systems have been incorporated into the mainstream healthcare in order to solve particular problems. These systems, however, are mostly interdependent in terms of healthcare data and processes - that is, requiring high-level of interoperability mainly addressed by the healthcare initiatives through their standards [70, 71, 73]. We identify the following groups by classifying the health information systems according to their functionalities [74]:

- *Hospital Information Systems:* Hospital information systems (HIS) are just one, but crucial, instance of health information systems, to manage the administrative tasks and information flow within a hospital. The aim of hospital information systems is to contribute to a high-quality, efficient patient care and hospital management. The relevance of HISs for high-level quality of care is obvious, as without having appropriate access to relevant data, practically no decisions on diagnostic, therapeutic or other procedures can be made. Therefore, we can recognize the relevance of systematically processing data, information and knowledge for the quality and efficiency of healthcare.
- *Clinical Systems:* These systems mainly involve the administration of direct patient care using ICT to be used by general practitioners, pharmacists and dentists. In order to fulfill the needs of specific departments, they offer complex functionalities and information management capabilities. Various kinds of e-health systems such as Radiology Information Systems (RIS), Laboratory Information Systems (LIS), Cardiology Information Systems (CIS), Pathology Information Systems (PIS), E-Prescription and PACS can all be included in this category.
- *Health Information Portals:* These applications provide health-related information for patients and health professionals, and additionally they may provide possibilities for consultation or for buying pharmaceuticals or other health-related products.

- *Home-care Systems:* Systems that are used to deliver care services via telecommunication or wireless technologies to the patient at home. Examples of such systems are “remote vital signs monitoring systems” that enable the patient to receive targeted treatment and medication without the need to visit an outpatient clinic or occupy a hospital bed. These kinds of systems are particularly well developed in diabetes medicine, hypertension management, asthma monitoring and home dialysis.
- *Clinical Decision Support Systems:* Clinical decision support systems form a significant part of the field of clinical knowledge management technologies through their capacity to support the clinical process and use of knowledge, from diagnosis and investigation through treatment and long-term care. They can be defined as "active knowledge systems which use two or more items of patient data to generate case-specific advice". They are typically designed to integrate a medical knowledge base, patient data and an inference engine to generate case specific advice.
- *Electronic Healthcare Records (EHR) Systems:* Identical electronic patient information should not be stored redundantly. Unfortunately, at the present time, the same patient information tends to exist in many different forms and in many different locations. EHR systems mainly aim to collect, store, and maintain the healthcare data. This data can then be abstracted, reformatted, and rationally organized to support other e-health systems for their infrastructural and informational needs. EHR (also the central repository for patient information updates, further data analysis, and privacy controls) represents a major resource to fulfill these needs. Implementation of reliable EHR will provide a convenient and easy way to access timely, relevant, and accurate information. Transformation of the traditional health care system into e-health care relies on transformation of the management of health information and health information flow. EHR may thus be considered the lifeblood of e-health care.
- *Hospital Management, Supply Ordering, and Electronic Claim Processing Systems:* Considering the hospitals as enterprises, various hospital

management systems also exist in order to manage the healthcare accounting and supply chain. Enterprise Resource Planning, Accounting, Booking systems can be considered in this category. These systems mainly interact with HISs in order to optimize the healthcare processes and eliminate paper-based transactions.

In a heterogeneous environment like healthcare, the standards are necessary for the integration and interoperability of these systems. The standards provides a basis and domain knowledge for creating healthcare information systems in terms of health and patient information, clinical knowledge and workflow, messaging, interfacing, knowledge and data representation, and security (e.g. data privacy, confidentiality, individual and organization identifiers). These standards include the messaging standards such as the Health Level 7 (HL7) [75], Digital Imaging and Communications in Medicine (DICOM) [76]; data representation standards such as the Continuity of Care Record (CCR) [77], HL7 Clinical Document Architecture (CDA) [78]; medical terminologies such as SNOMED [79] , LOINC [80]; clinical context management standards such as HL7 Clinical Context Management Specification (CCOW) [81]; and electronic healthcare standards such as CEN EN 13606 EHRcom[82] , and openEHR [83]. However, using these standards in the realization of modern healthcare information systems to meet manage the increasing complexity in terms of data types, functionality, user types and emerging technologies [72] is not a straightforward task. In order to handle this issue, an initiative, Integrating the Healthcare Enterprise (IHE) [84], is formed to stimulate the integration of the information systems that support modern healthcare institutions. Its main objective is to support the use of existing standards and provide technical frameworks for the implementation of established standards to achieve specific clinical goals. Each technical framework includes a number of integration profiles which are business processes offering a common language for vendors and healthcare professionals for the implementation of healthcare infrastructures to manage real-world scenarios. For example, Figure 38 illustrates a process flow of Radiology Scheduled Workflow integration profile [85] together with its actors and transactions in order to place a radiology order. Currently, the following technical frameworks are provided to specify profiles for various areas in a healthcare environment; and they are expanded annually, after a period of public review, and maintained regularly by the IHE Technical Committees:

- IT Infrastructure Technical Framework
- Cardiology Technical Framework
- Eye Care Technical Framework
- Laboratory Technical Framework
- Pathology Technical Framework
- Patient Care Coordination Technical Framework
- Patient Care Devices Technical Framework
- Radiation Oncology Technical Framework
- Radiology Technical Framework

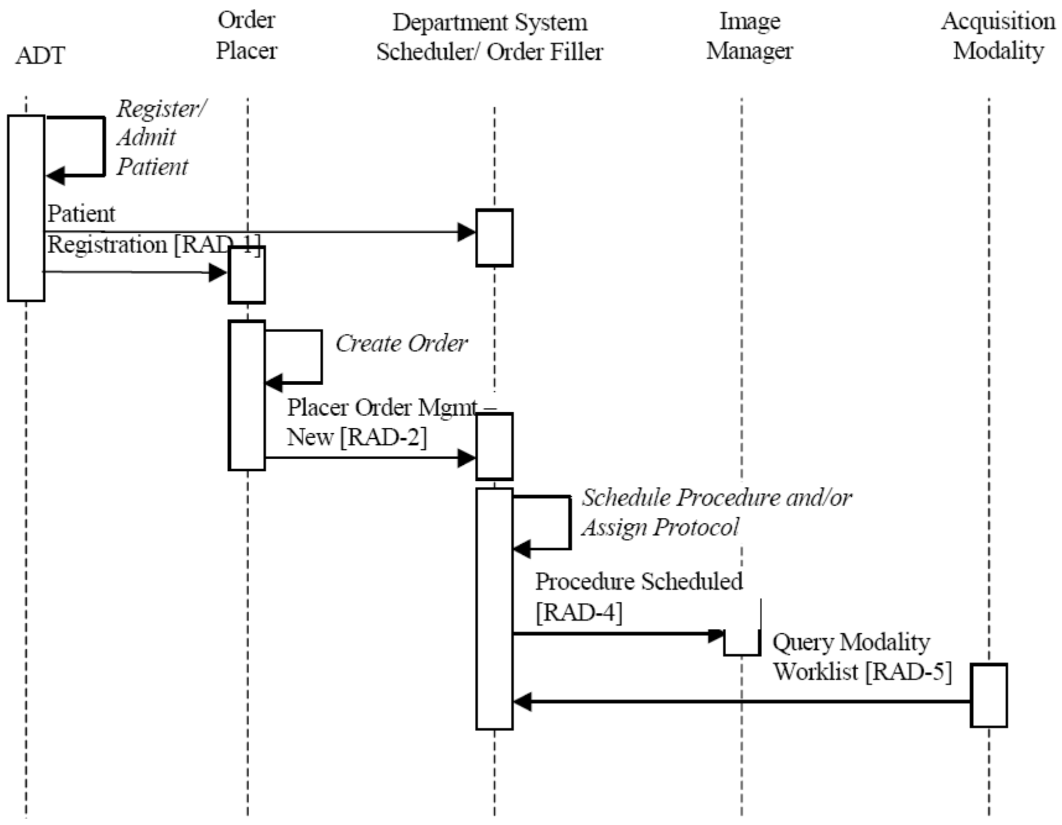

ADT
Order Placer
Department System Scheduler/ Order Filler
Image Manager
Acquisition Modality
Register/ Admit Patient
Patient Registration [RAD-1]
Create Order
Placer Order Mgmt – New [RAD-2]
Schedule Procedure and/or Assign Protocol
Procedure Scheduled [RAD-4]
Query Modality Worklist [RAD-5]

**Figure 38 An Example Process Flow of IHE Radiology Scheduled Workflow**

However, in realizing healthcare information systems which benefit from the existing standards as domain knowledge, there is a need to effectively coordinate several profiles to build the required system architecture. Although how to move from these highly specialized domain-specific profiles to service-oriented architecture is addressed in [15], a more feasible solution should be to consider the IHE profiles as domain-specific models and place them to the corresponding viewpoints in the architecture. This model-driven methodology enables a business-oriented design by directly enabling the system developers to create an architecture based on domain artifacts and facilitating the transition from the domain models to the platform through ontology mappings.

In order to utilize a semantic-based service-oriented design with the proposed methodology, we introduce a clinical mobile point-of-care (MPOC) application as a case study. It is a personalized application, deployed to a mobile device, which supports the physicians and nurses to perform their daily tasks at point-of-care in a motion intensive environment like hospitals. Data, such as patient records, clinical information, laboratory results, and diagnoses are also dynamic and should be accessed, updated and delivered to the place wherever it's needed based on the clinical workflows. The solution incorporates integrated healthcare information system services and uses the standards for the interoperability. Some of the essential tasks to be performed through the application can be stated as follows:

- Accurate access to patient identity and demographics
- Current episode information of the patient
- Summary of patient medical history
- Ordering Tests / Accessing Lab Results, Radiology and Pathology Reports
- RSS feeds from LIS, RIS when results are available
- Access to recent radiological images of the patient
- Viewing and updating of patient status, vital signs and diagnoses
- Placing of nursing orders

- Accessing Prescriptions
- Hospital / Clinic announcements as alerts or RSS feeds
- Accessing Drug Information
- Issuing Prescriptions

The underlying motivation for the MPOC application is to demonstrate and measure the benefits of utilizing services within a hospital. It aims to improve timely decision-making by making patient information, diagnostic data and expert decision support instantly accessible at the point-of-care. It is also intended to eliminate manual process barriers and multiple human interventions in patient care, and reduce medication errors by making use of right and accurate information access at the right place and at the right time. The study aims to provide the healthcare personnel a seamless environment to utilize IT services in order to offer more time to provide medical services. SOA is a good fit to realize such architecture in heterogeneous healthcare environments [86].

## 6.2. Engineering a Service-Oriented Clinical MPOC Application Architecture

Designing the MPOC system architecture based on service-oriented architecture starts with the specification of the system and the identification of the stakeholders in the architecture ontology. Besides the system architect, many people can be interested in the construction of the system. In our case, the clinicians, nurses, medical informatics experts can be stated as other stakeholders in the systems. As depicted in Figure 27, we derive the overall system architecture starting from the stakeholders' concerns. These concerns are, then, addressed by one or more features specified in the feature model of the system in order to enable a direct mapping from the problem domain to the architecture. Based on the problem description given above, a list of concerns and the corresponding features are specified in Table 2.

**Table 2 Concerns addressed by features**

| Concern | Feature(Type) |
|---|---|
| Create a MPOC System Architecture | MPOC System (Root) |
| Manage Patient Information | Patient (Feature), Get/Update Patient Demographics (Feature), ID Management (Feature), Get History (Feature) |
| Retrieve Patient Episodes | Episode (Feature) |
| Enable Radiology Orders, Appointments and Result Access | Radiology (Feature), Rad. Order Placing (Feature), Rad. Appointment Booking (Feature) |
| Enable Laboratory Orders, Appointments and Result Access Results | Laboratory (Feature), Lab. Order Placing (Feature), Lab. Appointment Booking (Feature) |
| Issue Prescriptions | Prescription (Feature), Submit Prescription (Feature) |
| Access Drug Information | Get Drug Info (Feature) |
| Manage RSS Feeds | RSS (Feature), Hospital RSS (Feature), Portal RSS (Feature) |

The domain analysis process starts with the specification of the concerns and identifying the concern-feature mappings. These features are organized in a feature model by specifying a hierarchy among the features, and constraints such as cardinalities. A feature model with the essential functionalities can be drawn as depicted in Figure 39. This model represents the result of the domain analysis process.

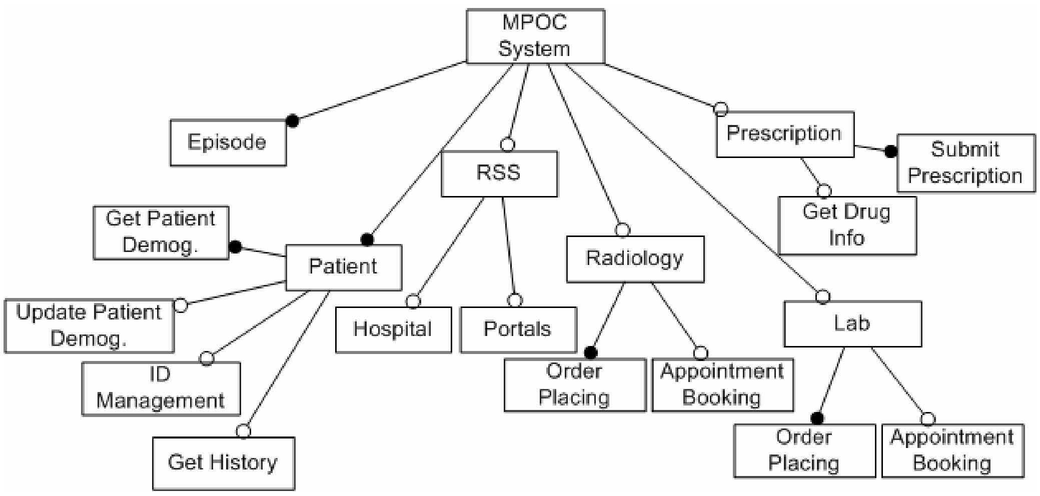


**Figure 39 Feature Model for Clinical Mobile Point-of-Care Application**

The specification of the MPOC architecture continues by realizing the system features with the viewpoints in the architecture ontology. Various kinds of viewpoints can be specified for different features. Viewpoints are basic architectural elements that encapsulate the other architecture components such as models and mappings. The identification of the viewpoints mostly depends on the architectural decisions to be made such as the chosen service-oriented technologies, platforms, existing architecture and other abilities of the development team. Assuming all the existing services from other systems in the healthcare setting is published to a UDDI registry, the basic features such as retrieving or updating the patient data, for example, can be achieved by consuming HIS services providing the required functionality. Therefore, a service discovery meta-model to allow the application developers specify the services to be discovered during application engineering are described as an element within a *UDDI Discovery Viewpoint* inherited from the *Discovery* viewpoint in the architecture ontology. The class hierarchy of the viewpoint and UDDI meta-model with the association between them is illustrated in Figure 40.

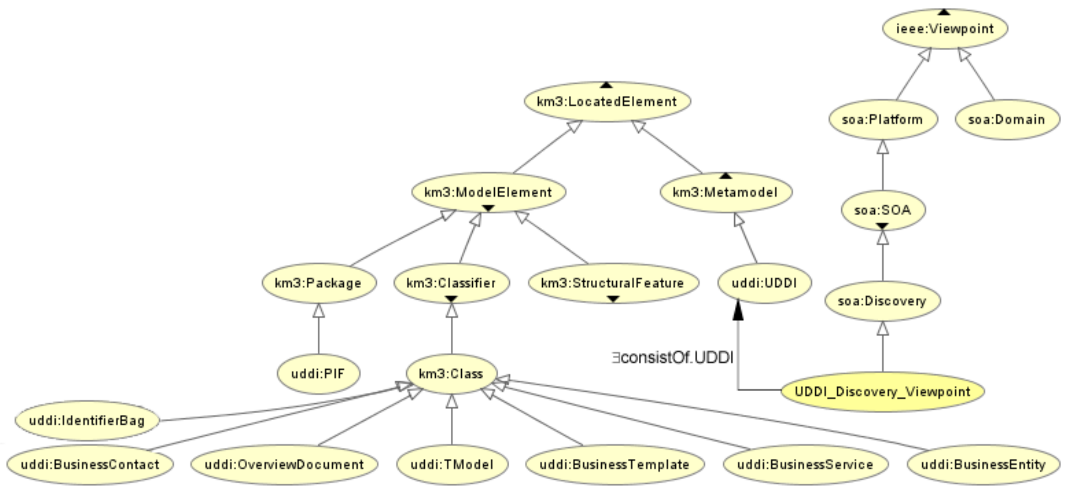


**Figure 40 UDDI Meta-model Classes and Viewpoint**

Although realizing some features in MPOC architecture can be straightforward, others may require a more complex utilization of models in different levels of abstractions together with mappings among these levels. As indicated in [15], modeling a radiology order process, for instance, requires the interaction with many systems as well as considering various standard profiles for interoperability. In addition, the modeling of the process should be achieved by using domain-specific models in order to better capture the requirements.

In order to achieve this, a multi-level modeling structure is needed in the viewpoint that realizes the Radiology services in the application. Based on the model description and mapping mechanisms presented in Chapter 5, it is possible to create a modeling structure as illustrated in Figure 41. This will allow the application developers of MPOC system to use the domain standards and domain-specific models to model the radiology ordering processes and enable a stepwise transition from the domain to the service-oriented technologies and platforms.

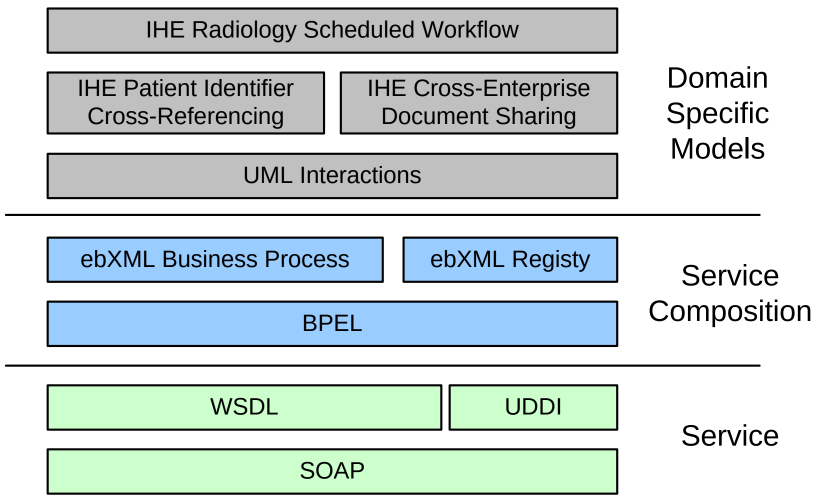


**Figure 41 Models and Levels in Radiology Service Viewpoint**

IHE Profiles utilizes the UML 2.0 Interaction Diagrams [87] as a platform independent models for specifying the healthcare processes. Therefore, we start the model specification process by introducing the UML 2.0 interaction model as illustrated in Figure 42. The Interaction class of this package encapsulates other elements that mainly define the interaction. The most visible aspects of the interaction are the lifelines representing the interacting parties and the messages exchanged between them. A sequence of *EventOccurences* is also used to represent the trace of the messages in the interaction. By using the interaction model, one can define several different types of interaction diagrams, including the sequence diagrams as it is the case for IHE Profiles.

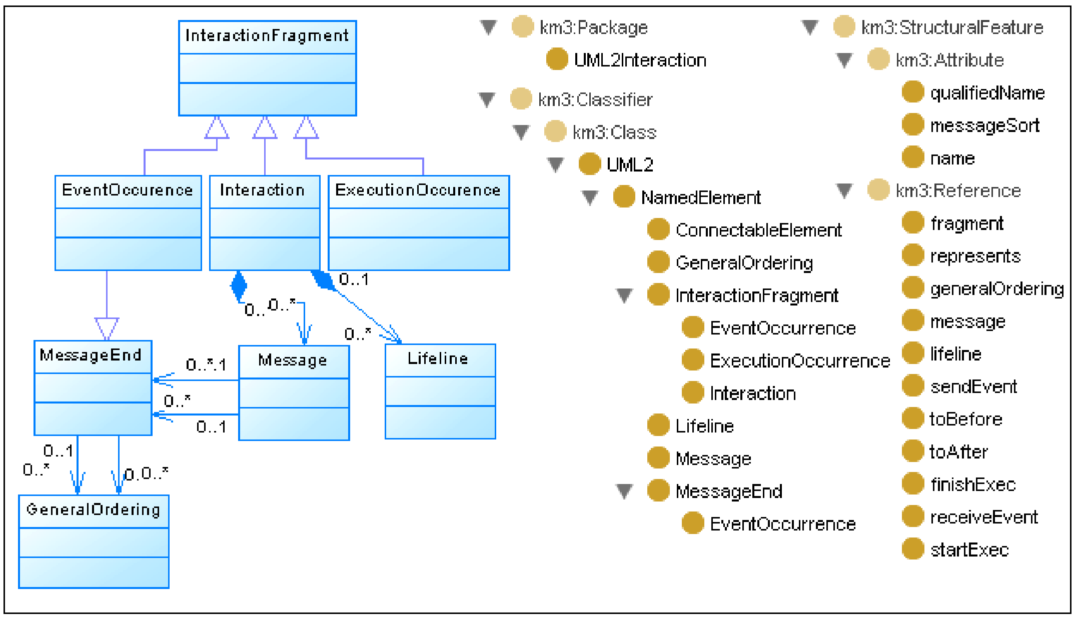


**Figure 42 The portion of UML 2.0 Meta-model around Interaction with the classes created by extending KM3 ontology**

IHE Profiles extends the UML 2.0 Interactions by defining the common actors and transactions available in a healthcare setting. Although each profile separately specifies its own actors and transactions, they need to be used in collaboration in order to meet the requirements of the healthcare systems [15]. For example, in our MPOC application, three profiles, namely Radiology Scheduled Workflow Profile, Patient Identifier Cross-Referencing Profile and Cross-Enterprise Document Sharing Profile, are used to specify the process and information flow with the radiology department, to transmit the patient identity information between departments, and to manage the sharing of healthcare documents, respectively.

Therefore, we complete our domain-specific model specification by extending the interaction model with the IHE actors and transactions. Each IHE actor is defined as a subclass of *Lifeline* class, while the IHE transaction extends the *Message* with specific constraints. For example, Patient Identifier Cross-Referencing Profile specifies a number of actors (Table 3) and transactions (Table 4) which can be defined in the model as shown in Figure 43. Therefore, the MPOC application queries the Cross-Reference Manager in order to obtain the patient identifiers before placing a radiology order by using the appropriate transactions.

**Table 3 IHE Patient Identifier Cross-Referencing Profile Actors**

| Actor | Description |
|---|---|
| Identity Source | Provides notification to the Cross-reference Manager for any patient identification related events |
| Cross Reference Manager | Manages the cross-referencing of patient identifiers across various domains |
| Cross Reference Consumer | Uses patient identifiers provided by the Cross-reference Manager to track patient identity |

**Table 4 IHE Patient Identifier Cross-Referencing Profile Transactions**

| Transaction | Description |
|---|---|
| Patient Identity Feed | Communicates patient information, including corroborating demographic data, after a patient's identity is established |
| Query | Involves a request by the Cross-Reference Consumer Actor for a list of patient identifiers that correspond to a patient identifier known by the consumer |
| Update Notification | Involves the Cross-reference Manager Actor providing notification of updates to cross-reference associations |

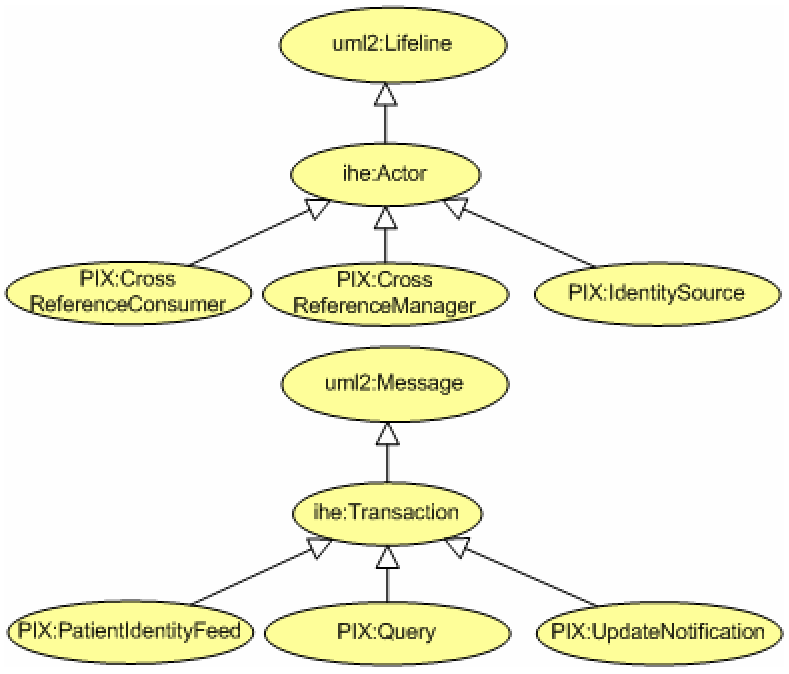
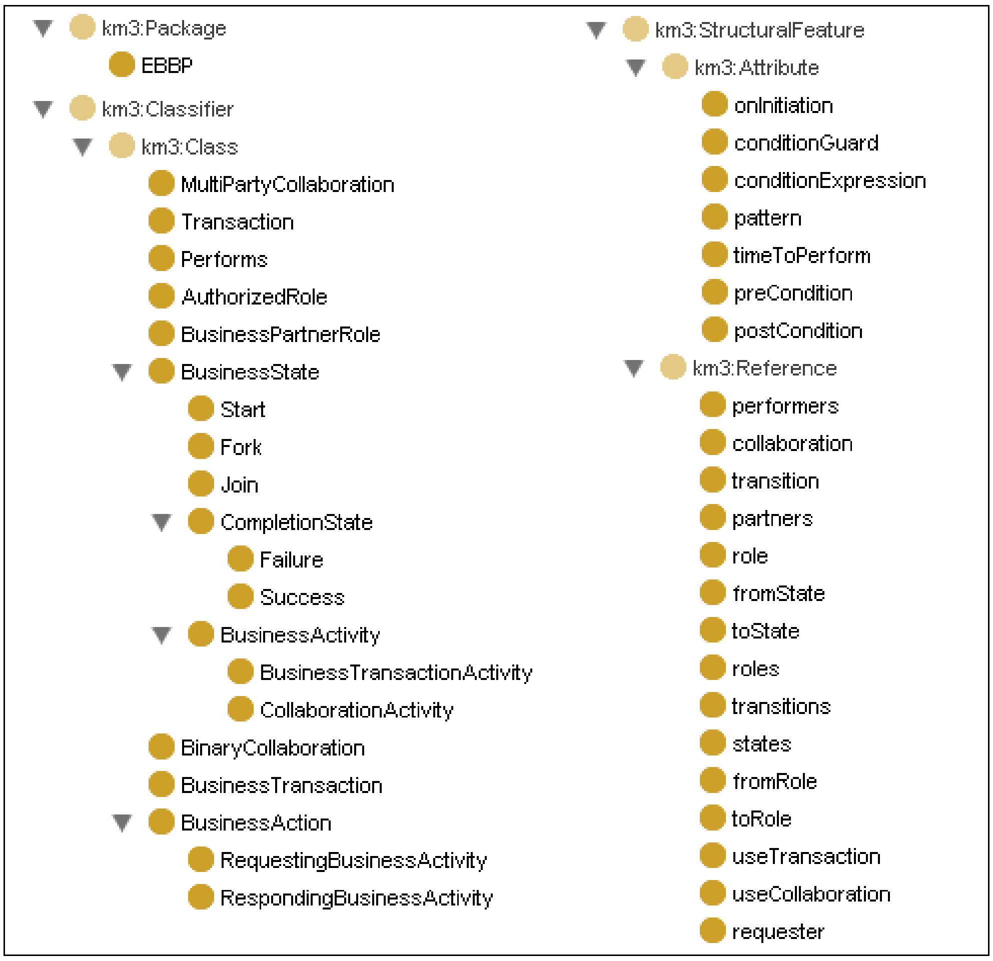


**Figure 43 IHE Patient Identifier Cross-Referencing Profile Classes**

Additionally, for the mapping of the domain-specific model to the service-oriented platform, the service composition models should be defined. We capture the ebXML Business Process (ebBP) schema as an ontology in order to represent the more abstract models in as ebXML choreographies (Figure 44). This model together with the other domain-specific and service-oriented models is included in the Radiology Service Viewpoint in the architecture description.

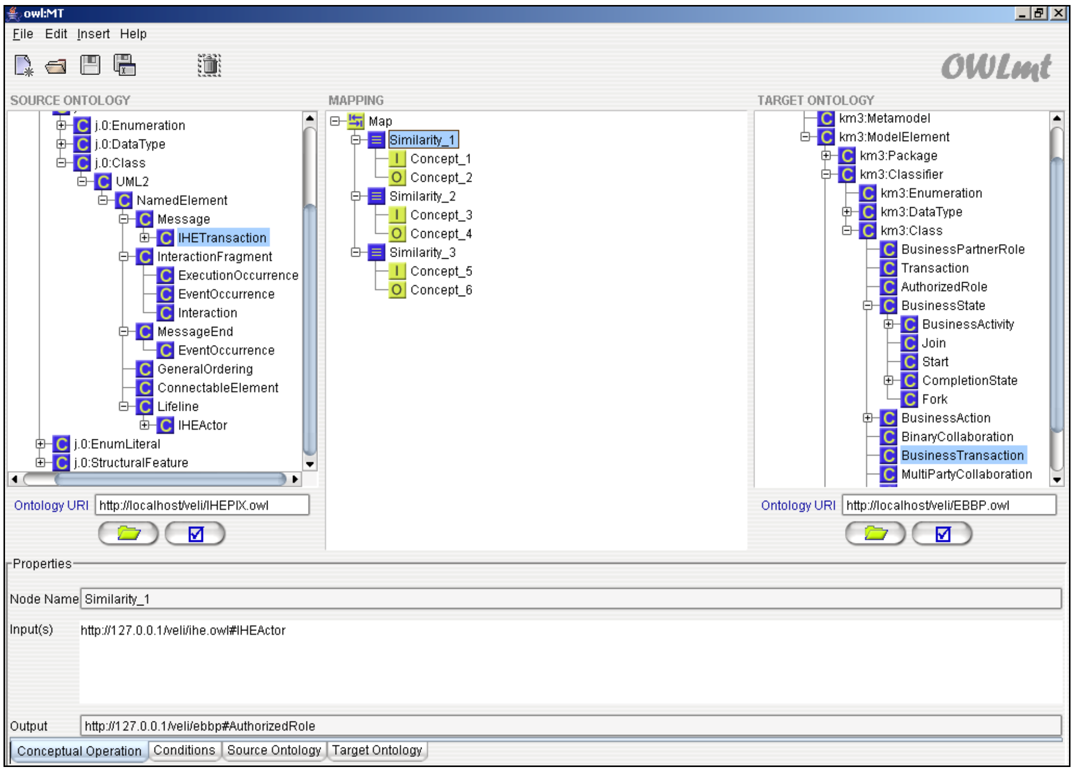


- km3:Package
  - EBBP
- km3:Classifier
  - km3:Class
    - MultiPartyCollaboration
    - Transaction
    - Performs
    - AuthorizedRole
    - BusinessPartnerRole
    - BusinessState
      - Start
      - Fork
      - Join
      - CompletionState
        - Failure
        - Success
      - BusinessActivity
        - BusinessTransactionActivity
        - CollaborationActivity
    - BinaryCollaboration
    - BusinessTransaction
    - BusinessAction
      - RequestingBusinessActivity
      - RespondingBusinessActivity
- km3:StructuralFeature
  - km3:Attribute
    - onInitiation
    - conditionGuard
    - conditionExpression
    - pattern
    - timeToPerform
    - preCondition
    - postCondition
  - km3:Reference
    - performers
    - collaboration
    - transition
    - partners
    - role
    - fromState
    - toState
    - roles
    - transitions
    - states
    - fromRole
    - toRole
    - useTransaction
    - useCollaboration
    - requester

**Figure 44 ebXML Business Process classes defined by extending KM3 ontology**

With the aim of realizing our domain-specific models over SOA, a transformation from the IHE-based models to the ebBP should also be defined within the Radiology Service viewpoint of the architecture. Although a direct mapping from IHE profiles to ebBP has been achieved in [15], this presents a number of limitations from a software engineering perspective. Since it eliminates the highly specialized models obtained from domain knowledge, it resembles a black-box modeling approach involving just one model specification (ebBP schema) to create the architecture. More importantly, it requires the specification of the domain with the particular SOA model. In order to manage this, we provide two different levels of models and specify the transformation from one to another through ontology mapping in OWLmt.

The mapping definition is specified at meta-model level by loading the ontologies of two model specifications into the OWLmt as shown in Figure 45. Firstly, the conceptual similarities between the model elements are specified in order to relate the ontologies at conceptual level. For this purpose, we identify the similarities at top level as shown in Table 5.



Ontology URI http://localhost/veli/IHEPIX.owl

Ontology URI http://localhost/veli/EBBP.owl

Node Name Similarity_1

Input(s) http://127.0.0.1/veli/ihe.owl#IHEActor

Output http://127.0.0.1/veli/ebbp#AuthorizedRole

**Figure 45 Mapping Definition in OWLmt between IHE/UML and ebBP Meta-models**

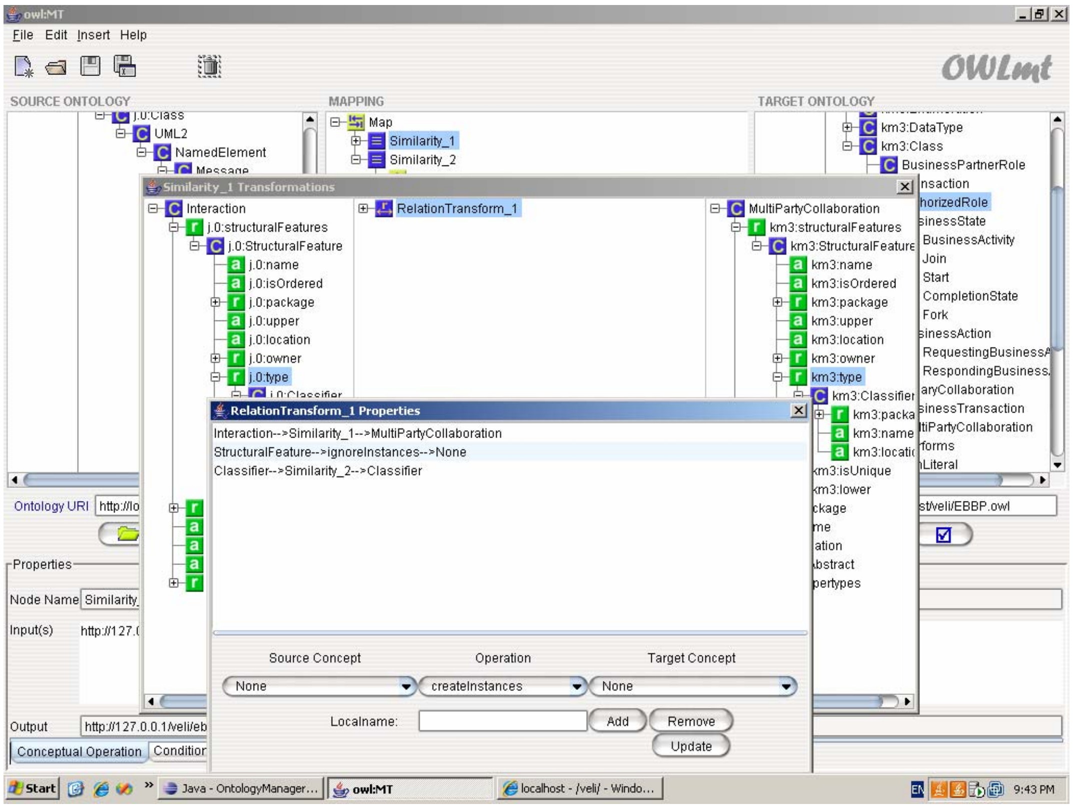
**Table 5 Top Level Similarities between IHE/UML and ebBP Concepts**

| IHE/UML | ebBP |
|---|---|
| IHE Actor, Lifeline | Authorized Role |
| IHE Transaction, Message | Business Transaction |
| Interaction | Multiparty Collaboration, Binary Collaboration |
| Message Argument | Business Document |

The similarities are captured in OWLmt with SimilarTo pattern to relate the classes of two ontologies. Every Interaction defined in the source model can be represented as business collaborations in ebBP, which can have a type of *MultipartyCollaboration* and *BinaryCollaboration*. A Business Collaboration consists of a set of roles that represent business partners. *AuthorizedRole* class, therefore, represents the IHE actors and *Lifelines* in ebBP side. Similarly, the IHE transactions and the *Messages* exchanged between the actors are mapped to the *BusinessTransaction* class. In addition, we can map any subclass of IHE Transaction to a particular pattern of *BusinessTransaction* which are specified in ebBP specification as Commercial Transaction, Notification, Information Distribution, Query/Response, Request/Confirm, Request/Response, or Data Exchange.

In addition to transforming the main model elements, the relations and attributes can be mapped to the corresponding elements in the target ontology. In order to achieve this, the object properties and data type properties are mapped within each mapping specification. For instance, for each Lifeline or Message contained within an Interaction, we need to create the required associations among the MultipartyCollaboration, AuthorizedRoles and Business Transactions. The OWLmt GUI to define these mappings is also shown in Figure 46.

Once the mappings are defined between the meta-models, they are also included within the corresponding architectural viewpoints in order to complete the architecture specification process. During the application engineering phase, the developers can model the application based on the meta-model specifications and the business requirements of a particular healthcare organization. The mapping definitions are also executed to convert these domain-specific models into a SOA. We consider that IT vendors or enterprises to choose from various service-oriented technologies and platforms; to construct an architecture once for a particular product type by specifying the architecture ontology, meta-models and mappings; to implement tools and editors for the selected models to ease the development; and to use conventional software engineering methodologies such as agile methodology or unified process for application development.



```
Interaction-->Similarity_1-->MultiPartyCollaboration
StructuralFeature-->ignoreInstances-->None
Classifier-->Similarity_2-->Classifier
```

**Figure 46 Specifying Mappings among Class Properties**

# 7. CONCLUSION

In this research, we tried to show how to exploit the semantic Web technologies to create SOAs in a model-driven way. The use of ontologies for the representation of the architecture description enables a way for metadata management by exploiting the existing technologies. The ontologies to represent architecture, features and the models provides better means to clearly specify the connections among various elements that are supposed to co-exist in the architecture.

By extending IEEE architecture standard, we recommend an architecture description ontology that enables a transition from stakeholders' concerns to the model specifications. This ontology refers to other sub-ontologies that define the architectural elements in detail to let the architects define the system specification within a two-phase methodology. This is a required step for SOAs since SOA involves a number of technologies and platforms to be precisely aligned prior to application engineering.

The utilization of the ontologies to facilitate a model-driven design is also another contribution presented in this thesis. The representation of KM3 meta-meta model in OWL facilitates a bridge between the AMMA modeling space and the Semantic Web modeling space. This allows us to utilize other semantic Web technologies such as ontology repositories, query languages and ontology mapping to manage the models and model transformations. Although ontology mapping tool, OWLmt, is previously employed to semantically mediate healthcare messages and electronic healthcare records, its use for model transformations provides a new perspective for enterprise system development. A comparison between ATL and OWLmt is given in order to support this idea by showing the features of the proposed approach. The case study indicates that approach can be useful in defining multi-level architectures and transition from domain-specific models to service-oriented technologies.

## 7.1. Future Work

Our analysis and research on service-oriented system engineering show that this is an evolving area and fertile ground for research. The following tracks of research are suggested:

- Service-oriented architectures provide a set of specifications and technologies built on top of the services in a loosely-coupled way leading to the separation of concerns in different models. The model composition techniques can be developed to create a composition framework by facilitating the aspect-oriented programming on service-oriented runtime in order to cleanly modularize the crosscutting concerns like security, management, mediation as aspects.
- With the aim of enabling the exchange and reuse of the model specifications among the enterprises, a collaboration network can be provided to semantically query, discover and obtain the model specifications in a peer-to-peer network infrastructure for domain level cooperation.
- The effects of globalization are forcing the enterprises to explore ways to diversify and deliver software in a timely and productive manner. In order to achieve this, Software Product Lines (SPL) have emerged as one of the most promising software development paradigms over the last decade by utilizing a number of core assets in product-line architecture to create a set of software products for a particular domain. However, creating SPL architectures require upfront investment for core asset development and introduce complexities in terms of scoping, variability management, evolution and testing. Actually, if they are complemented with a domain engineering approach, current service-oriented technologies constitute a unique set of core assets in order to foster the enterprise level reuse and product-line development increasing the return-of-investment for enterprises. By utilizing a SPL approach, service-oriented domain-specific platforms can be created for the enterprises to build systems over SOA.

We believe that this track of research requires further attention and insights gained through such studies that will result in discovering new service-oriented system development methodologies.

# REFERENCES

[1] A. Arsanjani, B. Hailpern, J. Martin, and P. L.Tarr, "Web Services: Promises and Compromises," IBM 2002.

[2] M. Brodie, C. Bussler, J. d. Brujin, T. Fahringer, D. Fensel, M. Hepp, H. Lausen, D. Roman, T. Strang, H. Werthner, and M. Zaremba, "Semantically Enabled ServiceOriented Architectures: A Manifesto and a Paradigm Shift in Computer Science," DERI 2005.

[3] W. B. Rouse, "A theory of enterprise transformation," *Systems Engineering* vol. 8, pp. 279 - 295, 2005.

[4] Z. Stojanovic and A. Dahanayake, *Service-Oriented Software System Engineering: Challenges and Practices*. Hershey-London: Idea Group Publishing, 2005.

[5] WSMO-WSMX, "Web Service Modelling Execution Environment(WSMX)", Last access date: July 2007, from http://www.wsmx.org/

[6] Apache-Tuscany, "Apache Tuscany Project", Last access date: July 2007, from http://incubator.apache.org/tuscany/home.html

[7] Apache-WS, "Apache Web Services Project", Last access date: July 2007, from http://ws.apache.org/

[8] Mule, "Mule Open Source Enterprise Service Bus", Last access date: July 2007, from http://mule.codehaus.org/

[9] ActiveBPEL, "ActiveBPEL Open Source WS-BPEL Engine Project", Last access date: July 2007, from http://www.active-endpoints.com/open-source-active-bpel-Intro.htm

[10] A. Colyer, G. Blair, and A. Rashid, "Managing Complexity In Middleware," in *The Second AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, 2003.

[11] P. Herzum and O. Sims, *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*: John Wiley & Sons, 2000.

[12] T. Margaria and B. Steffen, "Service Engineering:Linking Business and IT," *IEEE Computer*, vol. 39, pp. 45-55, 2006.

[13] G. Kotonya, J. Hutchinson, and B. Bloin, "A Method for Formulating and Architecting Component- and Service-Oriented Systems," in *Service-Oriented Software System Engineering: Challenges and Practices*, Z. Stojanovic and A. Dahanayake, Eds.: Idea Group Publishing, 2005, pp. 155-181.

[14] J. Zdravkovic, M. Henkel, and P. Johannesson, "Moving from Business to Technology with Service-Based Processes," *IEEE Internet Computing*, vol. 9, pp. 73-81, 2005.

[15] A. Dogac, V. Bicer, and A. Okcan, "Collaborative Business Process Support in IHE XDS through ebXML Business Processes " presented at International Conference on Data Engineering (ICDE), Atlanta, USA, 2006.

[16] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Kramer, "Service-Oriented Computing: A Research Roadmap," presented at Service Oriented Computing (SOC) 2005, Schloss Dagstuhl, Germany, 2006.
[17] OMG-BPMN, "Business Process Modeling Notation Specification," OMG 2006.
[18] M. Colombo, E. D. Nitto, M. D. Penta, D. Distante, and M. Zuccala, "Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems," presented at International Conference on Service Oriented Computing(ICSOC), Amsterdam, Netherlands, 2005.
[19] K. Czarnecki, M. Antkiewicz, and C. H. P. Kim, "Multi-Level Customization In Application Engineering: Developing mechanisms for mapping features to analysis models," *Communications of the ACM*, vol. 49, pp. 61-65, 2006.
[20] K. C. Kang, J. Lee, and P. Donohoe, "Feature Oriented Product Line Software Engineering: Principles and Guidelines," in *Domain Oriented Systems Development: Perspectives and Practices*: Taylor & Francis, 2003.
[21] IEEE, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," vol. IEEE Std 1471-2000: IEEE Architecture Working Group, 2000.
[22] J. Bosch, "Software Architecture: The Next Step," presented at European Workshop on Software Architecture, St Andrews, UK, 2004.
[23] F. Bachmann, L. Bass, J. Carriere, P. Clements, D. Garlan, J. Ivers, R. Nord, and R. Little, "Software Architecture Documentation in Practice: Documenting Architectural Layers," Carnegie Mellon University Software Engineering Institute 2000.
[24] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond* Addison-Wesley Professional, 2002.
[25] M. P. Papazoglou, "Web Services Technologies and Standards," *ACM Computing Surveys*, vol. Submitted for publication, 2006.
[26] A. P. Barros and M. Dumas, "The Rise of Web Service Ecosystems," *IEEE Computer*, vol. 8 pp. p31-37, 2006.
[27] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A Note on Distributed Computing," Sun Microsystems Laboratories, Mountain View, CA, USA 1994.
[28] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture", Last access date: from http://www.w3.org/TR/ws-arch/
[29] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1 ", Last access date: July 2007, from http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[30] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", Last access date: July 2007, from http://www.w3.org/TR/wsdl
[31] L. Clement, A. Hately, C. v. Riegen, and T. Rogers, "UDDI Version 3.0.2", Last access date: July 2007, from http://uddi.org/pubs/uddi_v3.htm

[32] M. P. Papazoglou and W.-J. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The International Journal on Very Large Data Bases (VLDB Journal)*, vol. 16 pp. 389 - 415, 2007.

[33] V. Bicer, C. Togay, and A. H. Dogru, "A Model-Driven Approach for Service-Centric System Development," presented at Integrated Design and Process Technology, IDPT-2007, Antalya, Turkey, 2007.

[34] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, pp. 46-52, 2003.

[35] F. Leymann, "Web Services Flow Language," IBM 2001.

[36] S. Thatte, "XLANG: Web Services for Business Process Design," Microsoft 2001.

[37] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services, Version 1.1," BEA, IBM, Microsoft, SAP, Siebel 2003.

[38] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto, "Web Services Choreography Description Language, Version 1.0", Last access date: July 2007, from http://www.w3.org/TR/ws-cdl-10/

[39] J.-J. Dubray, S. S. Amand, and M. J. Martin, "ebXML Business Process Specification Schema Technical Specification v2.0.4," OASIS Standard 2006.

[40] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, pp. 41-50, 2003.

[41] M. P.Papazoglou and W.-J. v. d. Heuvel, "Web Services Management: A Survey," *IEEE Internet Computing*, vol. 9, pp. 58-64, 2005.

[42] N. Catania, P. Kumar, B. Murray, H. Pourhedari, W. Vambenepe, and K. Wurster, "Web Services Management Framework - Overview Version 2.0," 2003.

[43] H. Kreger and T. Studwell, "Autonomic computing and Web Services Distributed Management," IBM, 2005.

[44] G. Muller, "Architectural Reasoning Explained," Embedded Systems Institute, Eindhoven, The Netherlands 2006.

[45] B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly," *IBM Systems Journal*, vol. 45, pp. 451-461, 2006.

[46] F. Jouault and J. Bézivin, "KM3: A DSL for Metamodel Specification," presented at 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy, 2006.

[47] K. Czarnecki and S. Helsen, "Feature-based Survey of Model Transformation Approaches," *IBM Systems Journal*, vol. 45, pp. 621-645, 2006.

[48] OMG-MDA, "Model Driven Architecture", Last access date: July 2007, from http://www.omg.org/mda/

[49] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*: Springer-Verlag, 2005.

[50] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez, "Modeling in the Large and Modeling in the Small," presented at European MDA Workshops: Foundations and Applications, MDAFA Twente, The Netherlands and Linköping, Sweden, 2003/2004.

[51] J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*: John Wiley & Sons 2004.

[52] F. Jouault and I. Kurtev, "Transforming Models with ATL," presented at MoDELS Satellite Events 2005, Montego Bay, Jamaica, 2005.

[53] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, pp. 34-43, 2001.

[54] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *International Journal of Human Computer Studies*, vol. 43, pp. 907-928, 1995.

[55] G. Klyne and J. J. Carroll, "Resource Description Framework (RDF): Concepts and Abstract Syntax", Last access date: July 2007, from http://www.w3.org/TR/rdf-concepts/

[56] D. Brickley and R. V. Guha, "Resource Description Framework (RDF) Schema Specification 1.0", Last access date: July 2007, from http://www.w3.org/TR/rdf-schema

[57] D. L. McGuinness and F. v. Harmelen, "OWL Web Ontology Language Overview ", Last access date: July 2007, from http://www.w3.org/TR/owl-features/

[58] J. d. Brujin, "Logics for the Semantic Web," DERI 2006.

[59] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The description logic handbook*: Cambridge University Press, 2003.

[60] J. Broekstra, A. Kampman, and F. v. Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," presented at International Semantic Web Conference(ISWC), Sardinia, Italy,, 2002.

[61] V. Bicer, "OWLmt: An OWL Mapping Tool - Technical Report," METU 2004.

[62] XQuery-XPath-Functions, "XQuery 1.0 and XPath 2.0 Functions and operators", Last access date: July 2007, from http://www.w3.org/TR/xpath-functions/

[63] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison-Wesley, 2000.

[64] H.-J. Happel and S. Seedorf, "Applications of Ontologies in Software Engineering," presented at 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), Athens, GA, U.S.A., 2006.

[65] P. Kruchten, H. Obbink, and J. Stafford, "The Past, Present and Future of Software Architecture," *IEEE Software*, vol. 23, pp. 22-30, 2006.

[66] G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel, "Product Line Analysis: A Practical Introduction," Carnegie Mellon Software Engineering Institute 2001.

[67] V. Bicer and C. Togay, "Representing Feature Models with Semantic Web Ontologies," presented at First Turkish Software Architecture Design Conference, Istanbul, Turkey, 2006.

[68] H. Wang, L. Y. Fang, J. Sun, H. Zhang, and J. Pan, "A Semantic Web Approach to Feature Modeling and Verification," presented at Workshop on Semantic Web Enabled Software Engineering (SWESE 2005), Galway, Ireland, 2005.

[69] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing Cardinality-based Feature Models and their Specialization," *Software Process Improvement and Practice, John Wiley & Sons*, vol. 10, pp. 7-29, 2005.

[70] V. Bicer, G. B. Laleci, A. Dogac, and Y. Kabak, "Artemis Message Exchange Framework: Semantic Interoperability of Exchanged Messages in the Healthcare Domain," *ACM Sigmod Record*, vol. 34, 2005.

[71] V. Bicer, O. Kilic, A. Dogac, and G. B. Laleci, "Archetype-based Semantic Interoperability of Web Service Messages in the Healthcare Domain," *International Journal on Semantic Web and Information Systems*, vol. 1, pp. 1-22, 2005.

[72] R. Haux, "Health Information Systems - Past, Present, Future," *International Journal of Medical Informatics*, vol. 75, pp. 268-281, 2006.

[73] V. Bicer, G. B. Laleci, A. Dogac, and Y. Kabak, "Providing Semantic Interoperability in the Healthcare Domain through Ontology Mapping," presented at eChallenges 2005, Ljubljana, Slovenia, 2005.

[74] J. Tan, *E-Health Care Information Systems*: Jossey-Bass, 2005.

[75] HL7, "Health Level 7 (HL7)", Last access date: July 2007, from http://www.hl7.org/

[76] DICOM, "Digital Imaging and Communications in Medicine (DICOM)", Last access date: July 2007, from http://medical.nema.org/

[77] ASTM-CCR, "ASTM The Continuity of Care Records(CCR)", Last access date: July 2007, from http://www.astm.org/COMMIT/E31_ConceptPaper.doc

[78] HL7-CDA-Release2.0, "The HL7 Version 3 Standard: Clinical Data Architecture," ANSI Standard, 2005.

[79] SNOMED., "The Systematized Nomenclature of Medicine(SNOMED) Clinical Terms."

[80] LOINC, "Logical Observation Identifiers Names and Codes (LOINC)", Last access date: July 2007, from http://www.loinc.org/

[81] HL7-CCOW, "HL7 Clinical Context Management Specification (CCOW)", Last access date: July 2007, from http://www.hl7.org.au/CCOW.htm

[82] CEN-PREN-13606-1, "Health informatics-Electronic health record communication-Part 1: Reference model. Draft European Standard for CEN Enquiry prEN 13606-1," European Committee for Standardization, Brussels, Belgium 2004.

[83] openEHR, "openEHR Community", Last access date: July 2007, from http://www.openehr.org/

[84] IHE, "Integrating the healthcare enterprise (IHE)", Last access date: July 2007, from http://www.ihe.net/

[85] IHE-Radiology, "IHE Radiology Technical Framework Rev. 7.0. Integration Profiles", Last access date: July 2007, from http://www.ihe.net/Technical_Framework/upload/ihe_tf_rev7.pdf

[86] E. D. Valle, D. Cerizza, V. Bicer, Y. Kabak, G. B. Laleci, and H. Lausen, "The Need for Semantic Web Service in the eHealth," presented at W3C workshop on Frameworks for Semantics in Web Services, Innsbruck, Austria, 2005.

[87] UML2.0-Superstructure, "Unified Modeling Language 2.0: Superstructure," OMG 2007.