Framework for Requirements-Driven System Design Automation

Ionut Cardei, Mihai Fonoage, and Ravi Shankar
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

***Abstract*— In this paper we present a framework for improving model-driven system design productivity with Requirements-Driven Design Automation (RDDA). The key to the proposed approach is to close the semantic gap between requirements, components and architecture by using compatible semantic models for describing product requirements and component capabilities, including constraints. An ontology-based representation language is designed that spans requirements for the application domain, the software design domain (UML meta-schema) and the component domain. Design automation is supported for architecture development by machine-based mapping of desired product/subsystem features and capabilities to library components and by synthesis and maintenance of UML design structure diagrams. The RDDA framework uses standards-based semantic web technologies and can be integrated with exiting modeling tools.**

I. Introduction

As part of the system development cycle, an iteration of a model-driven development process begins with requirements specification, where marketing specialists and product managers describe functional requirements, technical specifications, features and use cases in natural language, in a semi-formal format, such as Marketing Requirements Documents, Unified Modeling Language (UML) [8] or System Modeling Language (SysML) [7]. Following the requirements specification, system architects and engineers create a hardware/software architecture design that must fulfill all the requirements and satisfy any specified constraints. This design stage includes mapping product features, Quality of Service (QoS) constraints and behaviors to a component-based architecture. This product decomposition process involves QoS translation (from product to sub-product to component level), matching requirements/constraints to components, and component configuration. The implementation and deployment stages follow thereafter.

The transition from requirements to an architecture design is largely done manually with the help of modeling tools, such as Model Driven Architecture (MDA) UML editors. In most cases designers have available a considerable volume of pre-existing components and frameworks, from earlier projects or from third parties. With libraries holding hundreds of components, finding a configuration with components that matches all requirements and constraints without any conflicts could take considerable time and manpower, especially when requirements are updated frequently or what-if exploration and trade-off analyses are performed.

This paper presents a framework for Requirements-Driven Design Automation that aims to reduce the cost of system design by partially automating the process of architecture decomposition from requirements to existing library components. The framework can be applied to design of software and systems that use UML or SysML. This research is supported by the Motorola Corporation, from Plantation, Florida, under the *One Pass to Production* project and is currently ongoing at Florida Atlantic University's Center for Systems Integration.

The key to the proposed approach is to close the semantic gap between requirements, components and architecture by using compatible semantic models for describing both product requirements and component capabilities, including constraints. A domain-specific representation language is designed that spans the application domain (mobile applications, in our case), the software design domain (UML/SysML meta-schema) and the component domains. This language is used to represent ontologies, which are textual representations that capture the semantics of concepts common to product requirements and design modeling, and the relationships between them. The ontology (meta-model) for requirements specifications is based on the Semantic Web Ontology Web Language (OWL) [12]. It covers concepts for product requirements (features and structure), and a set of constraint checking rules. These rules permit consistency and completeness validation for requirements models before their use in architecture design. The RDDA ontology is expanded to cover knowledge representation for system architecture (UML and SysML diagrams) and for components (semantic annotations). In addition to specifications for product/subsystem features and capabilities, the RDDA ontology supports Quality of Service and system resource constraints, such as energy, CPU load, bandwidth, weight, and volume.

Design automation is supported for architecture development by component selection and design structure synthesis. Automated selection of components from libraries is useful when the number of candidate components is large or when there are many constraints that have to be met, including dependencies. Criteria for component selection include interfaces required and provided, implementation platform, capabilities and constraints that have to be satisfied. Selection criteria that cannot be represented in UML/SysML is described as OWL annotation metadata. Components that match the requirements and provide the necessary interfaces are pulled from the library to populate a structural UML diagram.

Synthesis of design structure diagrams takes the process further by producing new diagrams that can be edited by the user. It works bottom-up from existing structural models describing design relationships and derives feasible configurations represented as UML structural diagrams that satisfy the requirements.

The RDDA framework currently supports functional requirements expressed as required capabilities and constraints, such as symbolic elements (e.g. “supports GPS localization”) and numeric elements (e.g. “maximum latency is 10 s” or “cost between $10 and $20”.

The ontology representation also supports requirements tracing to design artifacts. This function is generally used by modeling software with requirements management capabilities to track designand implementation artifacts that are affected by changes to requirements.

The RDDA framework is designed to integrate with UML/SysML modeling tools compatible with OMG’s XML Metadata Interchange (XMI) format. Design models are translated to and from OWL specifications using Extensible Stylesheet Language Transformations (XSLT). This approach bypasses the limitations of the MDA’s Queries/Views/-Transformations (QVT), such as XMI-only conversion. At the core of the RDDA architecture is a reasoning framework built on top of the Jess Java rule engine [1] and the Jena Semantic Web toolset [3].

Due to space limitations, this paper presents the overall RDDA architecture and the methodology. Details on the specification language and on the rule-based processing will be published in a later article. This paper continues in the next section with a description of the RDDA framework. Section III describes the architecture of the Requirements-Driven Design Automation framework. Related work is described in section IV and section V summarizes this document with a discussion and conclusions.

II. The Requirements-Driven Design Automation Methodology

At the core of the requirements-driven design automationframework is the *OPP Design Language* (ODL), an OWL-based language that defines an ontology (taxonomy and properties) for describing semantic models for product requirements, component capabilities/constraints, and design artifacts (UML). In this context, ontologies encode domain-specific concepts and the relationships between them in a machine-processable format. An OWL format brings several advantages for knowledge representation in our project, from which the most relevant are that: a) it has a standardized XML-based encoding with wide tool support, b) the OWL DL dialect is supported by many reasoning engines and is decidable, c) OWL supports importing of other documents from the web or from a file system, and d) the XML encoding of OWL ontologies facilitates model transformation with XSLT [5].

In our project we target the development of mobile systems and applications. As the full scope of these domains is exceedingly large for the scope of this research, we develop prototype ontologies with limited scope that address a particular case study (an application for location-based services) and we provide a methodology for refining and extending ontologies by qualified personnel.

The RDDA methodology is illustrated in Figure 1. The input to the Requirements-Driven Design Automation methodology consists of: (a) requirement models specified in the formal ODL language capturing functional aspects, such as features, constraints and QoS. (b) component and UML classifier semantic annotation encoded in ODL describing capabilities and constraints, and (c) UML models encoded in XMI from which additional knowledge and relationships between components and classifiers are automatically extracted (e.g. dependencies, interfaces, associations).

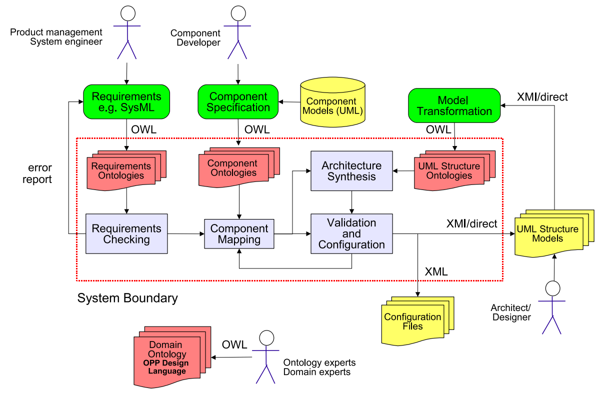

Figure 1. Requirements-Driven Design Automation methodology flow.

The RDDA methodology produces: (a) partial UML structural diagrams, (b) component configuration files (XML) and (c) reports on requirements consistency.

The main elements of the Requirements-Driven Design Automation methodology are listed below.

The OPP Design Language Definition

This language is defined initially by application domain experts and ontology experts. The ODL language consists of a series of OWL ontologies that covers the domains of application requirements, component capabilities and constraints, and system design (UML). Details follow in a later section.

Requirements Specification

The requirements models must be expressed in ODL. The ODL requirements taxonomy defines terms for describing features, capabilities and various constraints. Currently, a requirement specification document must be hand-written or developed with an OWL modeling tool, such as Protégé. As this is tedious and requires OWL expertise, we investigate alternative methods that are more user-friendly, such as using SysML/UML2.0 modeling tools, such as Rhapsody from Telelogic, and exporting to an XMI format. Requirements specifications in UML/SysML can be translated from XMI to an ODL ontology using XSLT model transformation process, described later. Another approach is to use modelers that process natural language requirements documents. Such a tool should produce semantic descriptions for features, constraints and QoS.

Component Specification

In a mature stage of the product line development process a stable population of software/hardware components is maintained. UML components are stored in their native UML file format or in XMI. For legacy software/hardware components, wrapper UML models can be built. Components are tagged with metadata in ODL that describe their semantics in terms of capabilities and constraints that cannot be captured in UML. For new components that are either developed in-house or acquired from third-parties throughout the design process, such metadata descriptions must be edited.

Design Modeling

The design modeling step involves the system architect and software designers. Users analyze requirements and create software design models that satisfy them. The RDDA methodology assists this modeling step by helping with selecting components from libraries that satisfy the requirements, by synthesizing new UML structure diagrams and by creating feasible component configurations.

The functional components of the RDDA framework are:

Requirements Checking

This component validates consistency of the requirements specified in ODL. It checks for any contradictions and verifies whether elements are missing from an existing specification. Validated requirement ontologies and component specifications expressed in ODL statements are compiled to facts in a knowledge base (KB).

Component Mapping

Component specifications in ODL are loaded in a knowledge base (KB) and passed to a reasoning engine that attempts to answer queries for matching components with requirements. The result from this step are sets of feasible components and configurations.

Architecture Synthesis

Architecture synthesis involves automatic analysis of design models in form of system design ontologies compiled from UML diagrams. Based on requirements – features, QoS and constraints – an automated rule-based reasoning mechanism generates new instances and properties that “fill in” the new and updated models such that they satisfy the requirements. When this is not entirely

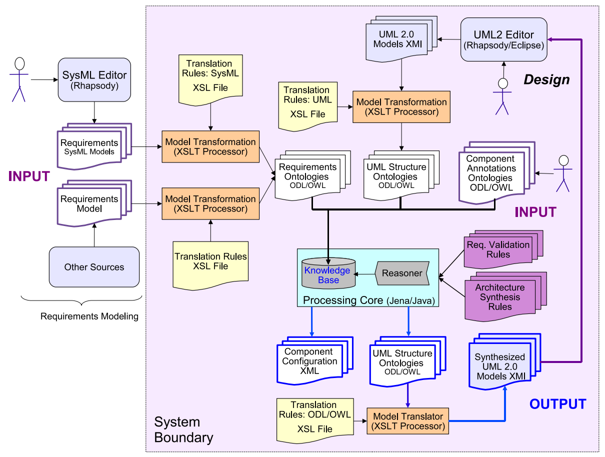
SysML Editor (Rhapsody)
Translation Rules: SysML XSL File
Translation Rules: UML XSL File
UML 2.0 Models XMI
UML2 Editor (Rhapsody/Eclipse)
Design
Model Transformation (XSLT Processor)
Requirements SysML Models
Model Transformation (XSLT Processor)
INPUT
Requirements Model
Model Transformation (XSLT Processor)
Requirements Ontologies ODL/OWL
UML Structure Ontologies ODL/OWL
Component Annotations Ontologies ODL/OWL
INPUT
Other Sources
Translation Rules XSL File
Knowledge Base
Reasoner
Processing Core (Jena/Java)
Req. Validation Rules
Architecture Synthesis Rules
Requirements Modeling
Component Configuration XML
UML Structure Ontologies ODL/OWL
Synthesized UML 2.0 Models XMI
OUTPUT
Translation Rules: ODL/OWL XSL File
Model Translator (XSLT Processor)
System Boundary

Figure 2. Requirements-Driven Design Automation framework architecture.

possible, the system will indicate the user the requirements that cannot be satisfied and the model components that are involved.

Validation and Configuration
The newly assembled design structures and the selected components are checked for consistency. Conflicting designs and component selections are reported to the user, marked and resubmitted to the Component Mapping process. Once the architecture designs are *feasible*, component parameters are derived and saved to runtime configuration files, if required.

The system architecture and the main methodology components are described in the next section.

III. RDDA Architecture

This section describes the high-level architecture of the RDDA framework, shown in Figure 2.

A. The OPP Design Language

The ODL must be able to express concepts from the domains of product requirements, component capabilities/constraints, and architecture models. The OWL ontology that describes the ODL vocabulary defines a taxonomy (OWL classes) and relationships between instances (OWL properties). Figure 3 shows a part of the ODL OWL class hierarchy.

The top part of the figure shows the hierarchy supporting the design metamodel concepts that are needed in ODL for describing components and UML structures. The *MetaModelConcept* classes are used to specify UML classes, interfaces, relationships, components and ports. Semantics are provided by properties binding to capabilities and constraints. The lower part of the figure shows a hierarchy derived from *RqConcept* class for describing requirements specifications. It features subsystem decomposition, feature sets, and defines a sub-hierarchy of features (functional and behavioral) and constraints: physical (weight, size), system resource limits (power, memory), and QoS (frame rate, latency). The ODL metamodel that forms the language to be used for RDDA is extensible and can pull in third-party ontologies through the OWL model import feature.

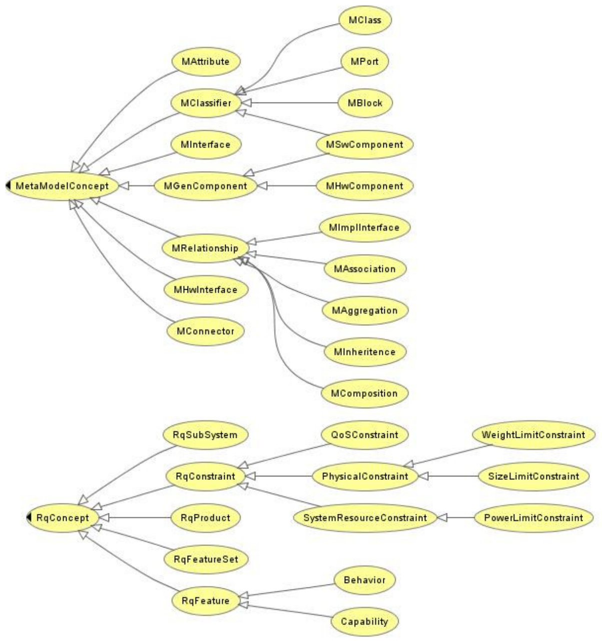

Figure 3. A snapshot of the OWL class hierarchy that forms the ODL metamodel.

1) The Requirements Domain: The Requirements Domain ontology defines concepts, relationships and properties necessary to describe system/application requirements. It defines also concepts for requirements tracking. Table 1 presents several concepts from this ontology.

2) The Design Domain: ODL ontologies describe UML and SysML structural models such as class diagrams, component diagrams, block diagrams and package diagrams. The ODL design domain includes concepts that can be found in these types of design diagrams, such as classes, ports, interfaces, blocks, components, packages, interfaces, relationships (generalization, aggregation, ...), attributes, methods, stereotypes, visibility.

Table 1
Concepts from the requirements ontology.

Concept	Purpose
RqProduct	The top-level system that is the subject of these requirements (system, application or a component).
RqSubsystem	A subsystem of the product design hierarchy.
RqFeature	A feature that is required to be supported by the designed system. Features can be capabilities or behaviors.
RqConstraint	A generic constraint that applies to a subsystem or component. Constraints can be physical constraints (weight, volume), QoS constraints (data rate, delay), and system resource constraints (CPU load, memory, bandwidth).
RqReqStmt	Represents the natural language text for one requirement statement.
RqVersion	Represents a requirements model version number.

3) The Component Annotation Domain: This ontology domain specifies semantic descriptions for component annotations, with focus on component capabilities, constraints, QoS and design elements, such as provided and required interfaces. In addition, the component annotation ontology covers platform-specific information, such as design level, hardware platform, OS, and programming language.

4) Example Requirements Specification: The following describes a brief sample requirement statement in English and its ODL encoding for a cell phone with a GPS positioning unit with certain QoS demands.

```
1.1 Product Motorola Phone 1300 has a GPS
    unit.
1.2 The GPS provides a location accuracy of
    minimum 10 m, a query response time of
    maximum 10 s.
```

The textual representation of these requirement statements is broken down and encoded in ODL facts with the $\langle subject, property, object\rangle$ OWL triple format. The ODL description must include definition of OWL class instances for document version, the product and subsystems involved, and required capabilities and constraints. Properties describe how these concepts are linked together. Statement 1.1 is encoded as follows:

```
<RqVersion rdf:ID="ReqVersion_v0.1"/>
  <RqReqStmt rdf:ID ="ReqStmt_1.1">
    <hasVersion rdf:resource="#ReqVers0.1"/>
    <rqTracks>
      <RqProduct rdf:ID="
           Product_Motorola_Phone_A1300">
        <hasSubsystem>
          <RqSubsystem rdf:ID="Subsys_GPS"/>
        </hasSubsystem>
      </RqProduct>
    </rqTracks>
    <reqStmtText>
      1.1 Product Motorola Phone 'A1300' has a
          GPS unit.
    </reqStmtText>
</RqReqStmt>
```

Requirement statement 1.2 defines QoS constraints for the GPS unit. The ODL specification includes *hasConstraint* references to OWL *RqConstraint* instances associated with accuracy and latency. The *RqFeature* property indicates a required capability. Their definition is not shown here for brevity.

```
<RqReqStmt rdf:ID ="ReqStmt_1.2">
  <hasVersion rdf:resource="#ReqVers0.1"/>
  <rqTracks>
    <RqSubsystem rdf:resource="#Subsys_GPS">
      <hasCapability rdf:resource="#
           GPS_Positioning"/>
      <hasConstraint rdf:resource="#
           QoS_minLocationAccuracy_10m"/>
      <hasConstraint rdf:resource="#
           QoS_maxQueryResponseTime_10s"/>
    </RqSubsystem>
  </rqTracks>
  <reqStmtText>
    1.2 The GPS provides a location accuracy
        of minimum 10 m, a query response
    time of maximum 10s.
  </reqStmtText>
</RqReqStmt>
```

For component selection and model synthesis, *RqConstraint* references from the requirements specification are matched against ODL annotations of library components.

B. Model Translation

UML and SysML models are exported to XMI portable specification format from a modeling tool, such as Rhapsody. The XMI format is based on XML and is converted to ODL to be compatible with the RDDA knowledge processing framework. The output from RDDA is converted back to XMI to be loaded into the UML/SysML modeling tool for further modeling use. For conversion we use the Extensible Stylesheet Language (XSL), an XML language for transforming and formatting XML-based documents standardized by the World Wide Web Consortium. Relevant parts of XSL that we need are XSL Transformations (XSLT) for converting XML documents and the XML Path Language (XPath), for navigating in XML documents.

The XSLT processor (SAXON) takes XML an input source file and an XSLT file with transformation rules, applies the transformation rules and generates a new XML file – an ODL document. The XSLT rules are called templates that contain match conditions and a set of instructions that are executed when match condition are satisfied. Statements written in the XPath language assist node matching and selection. An XSL transformation rule for UML interfaces from XMI code is shown below:

```
<xsl:template match="UML:Package/
    UML:Namespace.ownedElement/UML:Interface"
    >
  <odl:MSwInterface rdf:ID="{../../@name}.{
      @name}">
    <odl:hasName rdf:datatype="~xsd;string">
      <xsl:value-of select="@name"/>
    </odl:hasName>
    <odl:inPackage rdf:resource="#{../../
        @name}"/>
  </odl:MSwInterface>
</xsl:template>
```

The *xsl:template match* element defines a match condition for a UML interface in the input XMI file hat is converted to an *MSwInterface* ODL instance. *@name* will be replaced with the corresponding text from the XMI source.

There are several benefits for using XSLT for model transformation to/from ODL. XSLT is compatible with the OWL XML format and is data driven, not code driven. We only have to specify the transformation rules with matching patterns and the generated format, as opposed to actively traversing the XML document tree with DOM or SAX code. XSLT has a well defined XML data model that requires all compliant XSLT engines to parse XML in exactly the same way.

Knowledge Base and the Reasoner (Processing Core)
ODL ontologies encoding requirements, design models and component semantics are loaded in an internal knowledge base supported by the Jess framework [1] and the OWLJessKB semantic web library [9]. Jess is built on top of the Jena [3] semantic web Java framework. Jena provides persistence interfaces for both in-memory storage and database model storage. The reasoner component, provided by Jess, implements a rule-based inference engine that supports forward and backward chaining. Knowledge processing is guided by a set of rules, also loaded to the knowledge base. The Requirements Validation rules control the method for checking requirements model consistency. The Architecture Synthesis rules control the logic flow of architecture synthesis and component selection.

The output includes validation reports for requirements models, and UML models in ODL format that describe feasible design diagrams with component selection. In addition, an XML runtime configuration file is generated for components with a feasible parameter setup.

IV. Related Work

This section summarizes related work and provides pointers to more information. The work in [2] propose a Semantic Web approach for maintaining software, using the Resource Description Framework (RDF) for representing information related to software components, and the OWL for describing the relationships between those software components, metrics, tests, and requirements. Queries formulated in the SPARQL query language extract information about the state of the software system from the RDF knowledge base. The proposed solution deals only with software maintenance and does not address requirements specification/validation and UML design automation.

An approach for requirements processing based on ontologies is proposed in [4]. Requirements are checked for consistency and completeness by detecting conflicts and missing elements using inference rules, a method we also adopted. This work does not extend into design automation.

In [10] the authors present a software engineering tool called CASSANDRA that assists and guides developers through the software development process. It is implemented in WIN-PROLOG and has different interface agents and application agents that adapt to various external applications, such as UML CASE tools. CASSANDRA assists through the processes of Analysis, Design, Construction and Project Management. For each of these stages, an iterative approach is taken, resulting in the creation of a Specification, Architecture and Implementation. In the first two phases, a Domain Expert and Technology Expert are involved, for validation and verification purposes, but also for input on the technology part.

Automated synthesis of architecture design, addressed by RDDA , has a lot in common with dynamic composition of web services. Complex services can be assembled from atomic services in a similar way to how design artifacts (classes, components, interfaces) can be assembled in structural diagrams such as component diagrams, in order to achieve certain goals, such as providing a set of features while satisfying constraints. The use of Semantic Descriptions is proposed in [11] for finding, filtering, and integrating Web Services. Existing specification techniques, such as WSDL and UDDI, do not address the semantics of web services. The authors use OWL-S (formerly DAML-S) [13] specifications for semantic descriptions of service profiles, appropriate for automatic discovery and composition of services using a Prolog-backed OWL inference engine.

The work in [6] proposes a framework called Semantic Streams in which a user can take advantage of declarative statements to query a sensor network. The principles from the service and semantic domains are combined here to form a semantic services programming model where each service is a process that deduces semantic data about the world. These services are converted to rules with pre and postconditions and the inference engine uses backward chaining to match every element of the query with the post-condition of a service. In our approach, we follow a similar path for constraint validation and architecture synthesis. As in [11], composition is obtained by allowing processes that interpreting information to be wrapped up, forming semantically new applications.

V. Conclusions

In this paper we present a framework for improving the design productivity through automation. The main idea behind our approach is to describe product requirements and component semantics with a common language that supports automated reasoning on knowledge from application requirements, components and design models. The common specification is used to annotate library components with semantic descriptions of their capabilities and constraints. A rule-based reasoning engine performs two functions after validating the requirements specification: matching of requirements with components, and UML diagram synthesis.

Our approach supports only *static* description of capabilities/constraints. It does not address design artifact compatibility based on dynamic behavior. For instance, two components, one needing and one providing the same *LocationQuery* interface may differ in the order in which operations are invoked. The behavior specification for components and classes is given by UML behavioral diagrams (e.g. sequence and state machine diagrams). Currently, the diagram synthesis rules do not check for this behavior consistency when matching requirements with candidate design artifacts. We will look for a solution to this problem as it would improve the quality of the generated design.

The XMI-based method for model conversion introduces file consistency issues, as modeling tools do not always support the most recent versions of XMI. A better and more integrated solution is to plug the RDDA system directly in the modeling tool framework. For closed source tools this could be difficult. However, Eclipse has an open framework. We consider integrating our toolset with Eclipse as a plugin for an existing UML 2.0 modeler, such as Omondo, from www.omondo.com.

Another important issue is requirements specification. Describing product features, constraints and functional requirements in an OWL format is very tedious. Using OWL editing tools does not help much, since it still requires a deep knowledge of the domain ontologies. We will look into SysML for requirements specification.

Acknowledgments
This work could have not been possible without the generous support from Mr. Jaime Borras, Motorola Corporate Senior Fellow and Senior Vice President of Technology, and his colleagues from the Advanced Technology Group.

References

[1] Jess, the rule engine for Java. http://www.jessrules.com.
[2] D. Hyland-Wood, D. Carrington, and S. Kaplan. Enhancing software maintenance by using semantic web techniques. In *International Semantic Web Conference (ISWC)*, 2006.
[3] Jena. A semantic web framework for java. http://jena.sourceforge.net.
[4] Haruhiko Kaiya and Motoshi Saeki. Ontology based requirements analysis: Lightweight semantic processing approach. In *Fifth International Conference on Quality Software (QSIC 2005)*, 2005.
[5] Michael Kay. XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20/.
[6] J. Liu and F. Zhao. Towards semantic services for sensor-rich information systems. In *The 2nd International Conference on Broadband Networks*, pages 44–51, 2005.
[7] OMG. Omg systems modeling language. http://www.omgsysml.org/.
[8] OMG. The Unified Modeling Language. http://www.uml.org.
[9] OWLJessKB. OWLJessKB: A semantic web reasoning tool. http://edge.cs.drexel.edu/assemblies/software/owljesskb/.
[10] Markus Schacher. CASSANDRA: An automated software engineering coach. In *KnowGravity.com*, 2000.
[11] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web services: modeling, architecture and infrastructure workshop in ICEIS*, 2003.
[12] W3C. The ontology web language. http://www.w3.org/2004/OWL.
[13] W3C. OWL-S: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/.