Top-down Software Decomposition – An Approach for Component-based Design Automation

Whitepaper

Ionut Cardei

Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

07/21/2006

Abstract

In this whitepaper we summarize the objectives and the technical approach of the Top-down Software Decomposition project and we describe a methodology for improving the quality and reducing the costs of the design process. In our approach product requirements and component semantics are conceptualized in ontologies, providing semantic descriptions that are machine readable and can be processed for consistency checking and model synthesis through a process that involves machine reasoning.

This paper also provides a brief summary the main technologies involved in this project. The Model Driven Architecture provides the backdrop for a formal software development process and for visual modeling tools. We describe the UML and SysML modeling tools that are commonly used in the industry for requirements analysis and for software design. The main technologies from the semantic web research domain are covered, such as RDF, RDFS, OWL and reasoning engines.

1 Introduction

This project has focus on improving the architecture design quality, increasing productivity and reducing the cost of the development process. In this whitepaper we present our approach for automating component-based design through top-down decomposition and we introduce several supporting technologies and tools.

We begin by describing the problem we address and the motivation behind our project. As part of the system development cycle, a development iteration 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 MRDs, UML or SysML, or using requirements management tools such as DOORS

[23] or RequisitePro [21]. 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 the features, QoS constraints and behaviors to a component-based (and hierarchical) 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 UML modeling tools. In most cases designers have available a considerable volume of pre-existing components and frameworks, from earlier projects or from third parties. At an abstract level, building a component architecture from requirements is a search in a design space. With libraries holding hundreds of components, this search could take considerable time and manpower, especially when requirements are updated frequently or what-if exploration and trade-off analyses are performed.

This project aims to reduce the cost of system design by automating the process of architecture decomposition from existing component libraries. Specifically, our goal is to build a methodology that assists designers by matching product requirements with component capabilities to

- find component structures that satisfy requirements
- track components to requirements
- check requirements consistency
- produce component configurations.

This project uses industry standards, such as the Unified Modeling Language (UML) [19], the System Modeling Language (SysML) [16], and XML Metadata Interchange (XMI) [20] and complies with widely used UML modeling tools, such as Rhapsody [12].

1.1 Top-down Software Decomposition in the OPP Context (the *Big Picture*)

The Top-down Software Decomposition project is positioned in the overall OPP design process as bridging the requirements specification phase and the modeling/design phase, as shown in Fig. 1. Our methodology is used to describe requirements formally, and to define the capabilities and constraints of components. We will integrate our work with the results from the OPP "Specification Productivity" project, that addresses requirements specification for supporting model animation. The Top-down Software Decomposition project provides tools for processing (a) requirements in a formal representation and (b) component metadata to build and configure UML structural models that can be used directly in the design/modeling phases. The generated UML models will be either saved in XMI or automatically edited inside an UML modeling tool that supports a model access API (e.g. Eclipse or Rhapsody).

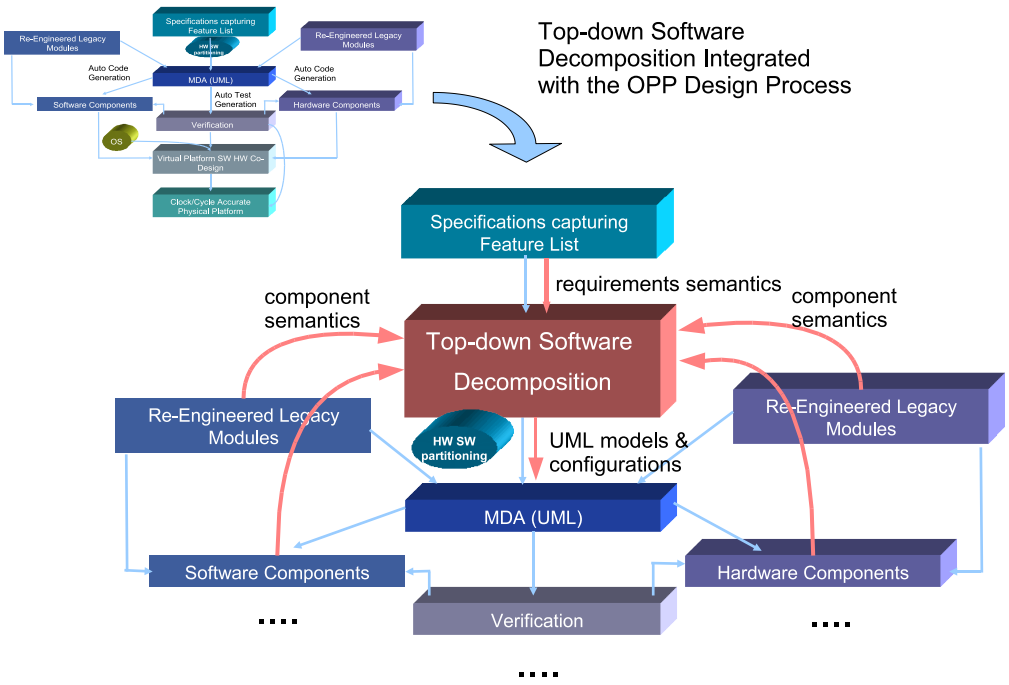

Figure 1: The Top-down Software Decomposition framework positioned in the overall OPP design process.

1.2 Ontology-based Technical Approach

The key to our solution is to close the semantic gap between requirements and components 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 (cell phones), the software design domain (UML/SysML metaschema) and the component domains. This OPP Design Language (ODL) is used to represent *ontologies*, textual representation (stored in files) that captures the semantic elements common to product requirements and design modeling, and provides the glue between them.

Ontologies encode domain-specific concepts and the relationships between them providing a vocabulary and a computerized specification of the meaning of terms used in the vocabulary. The ODL is defined using the Ontology Web Language (OWL), a logic-based language standardized by the W3C [27]. OWL has an XML encoding and has wide support in the semantic web community. Domain-specific ontologies written with ODL are also OWL ontologies and can be used for automated reasoning.

According to [11] ontologies can be very useful in software engineering projects where development is focused not just on one application, but on a family of projects from the same domain, as ontologies extend the idea of reuse from implementations (code) to the modeling level. This modeling approach is productive in *Model-Driven Development* (MDD), a modern paradigm that puts models at its core of its development process. For projects that cover new domains, ontologies must be developed in a new taxonomy framework to describe the

new concepts, properties, and relationships of the concern domains. The scope and generality of ontologies must be negotiated between stakeholders. For projects building on existing conceptual frameworks, existing ontologies can be used for laying out the analysis model. Definitions from the existing ontologies can be enriched and new elements can be added that are compatible with existing ontologies (condition called ontological commitment).

In the OPP project we target the development of mobile systems and applications, covering the domains of cell phone system architecture (hardware) and mobile applications (software). As the full scope of these domains is exceedingly large for the scope of the Top-down Software Decomposition project, we develop prototype ontologies with limited scope for both domains 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 input to our methodology consists of: (a) component and UML classifier metadata in ODL (for classes, interfaces, UML components) describing semantics, such as capabilities and constraints, (b) UML models in XMI from which additional knowledge and relationships between components and classifiers are automatically extracted (e.g. dependencies, interfaces, associations) and (c) requirements specified in the formal ODL language capturing mostly functional aspects, such as (sub)product features, constraints and QoS.
Note: a generic mechanism for extracting ODL ontologies from product requirements is beyond the scope of the Top-down Software Decomposition project. This is a challenging issue that needs further discussion with Motorola.

Our methodology produces: (a) partial UML structural diagrams, (b) component configuration files (XML) and (c) reports on requirements consistency. The overall design methodology described above is illustrated in Figure 2.
The main steps of the Top-down Software Decomposition methodology are:

ODL Definition

The ODL is the language for describing ontologies for the these problem domains: mobile applications, cell phone systems, hw/sw components and MOF metaschema (for components). This language is defined initially by the Top-down Software Decomposition team and extended later by domain experts to cover an increasing scope. The ODL language actually consists of a series of domain-specific OWL ontologies. Details on the ODL follow in section 3.1. Protegé [2] is an ontology modeling tool that will be used for defining ODL.

Requirements Specification

A recurring step in our methodology is application requirements specification by marketing and product management. In our vision, they would use a modeling tool that generates ODL ontologies encoding requirements. This can be achieved by compiling UML/SysML requirements models (usecase/activity/requirements/parametric/interaction) or by using domain-specific UML 2.0 profiles or Domain Specific Modeling tools that support requirements modeling and export ODL ontologies via *model transformation* (see section 4). In the limited scope of this project we assume requirements are expressed as ODL ontologies. Please read more in section 4 about requirements engineering with ontologies.

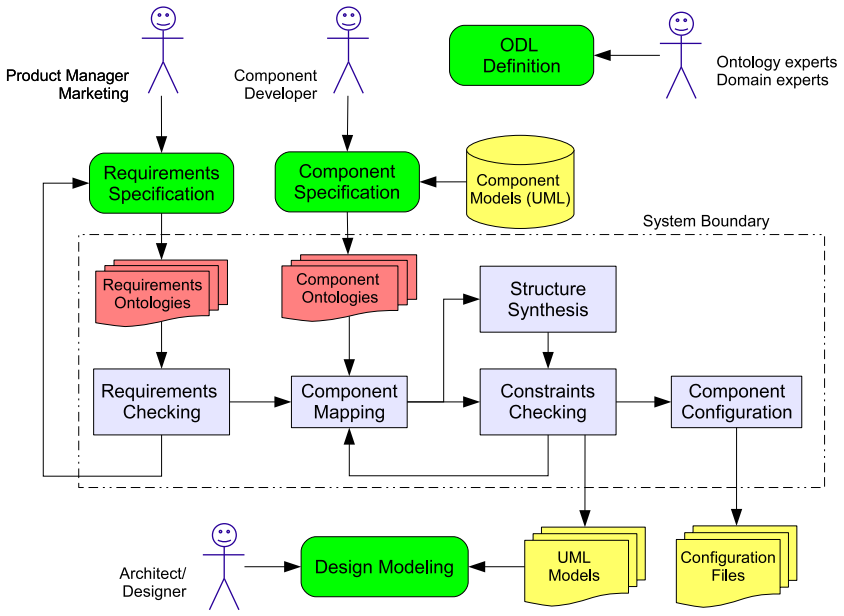

Figure 2: Top-down Software Decomposition methodology flow.

Component Specification

In the "evolved" stage of the OPP design process, it is assumed a stable population of sw/hw components already exists. UML components are stored in UML file (native) formats or in XMI. For legacy sw/hw components, wrapper UML models are built. Components are tagged with metadata in ODL that describe their semantics, in terms of capabilities and constraints that cannot be captured in UML and OCL. For new components that are either developed in-house or acquired from third-parties throughout the design process, such metadata descriptions must be edited. Initially metadata will be manually entered via a text editor or ontology tool (e.g. Protegé). In the OPP production flow, an integrated tool – a wizard – will be used to capture component metadata. The format and the semantics of the metadata permitted by the wizard tool are defined by ODL ontologies.

Design Modeling

The design modeling step involves the system architect and software designers. In this step these actors create software models from analyzing requirements. The Top-down Software Decomposition methodology assists this modeling step by helping with selecting components from libraries that satisfy the requirements, by synthesizing new UML structure models and by creating feasible component configurations. An example of how this automation works is given in section 3.1.

The functional components of the top-down decomposition system are:

Requirements Checking

This component validates consistency of the requirements specified in ODL. It checks for any contradictions and verifies whether any elements are missing from an existing specification. This is possible since ODL ontologies provide a metamodel specifying functional requirements, QoS requirements, and constraints.

Component Mapping

Validated requirement ontologies and component specifications expressed in ODL statements are compiled into facts and passed to a reasoning engine that attempts to match components with requirements. The result from this step are sets of components and *feasible* configurations that match the requirements, and reports of failures with descriptions of the mapping failure reasons.

Structure Synthesis

The previous component mapping step continues with this structure synthesis step where UML structure diagrams (class, block, component diagrams) are assembled to model higher level components that satisfy the requirements.

Constraints Checking

The newly assembled structures and the selected components are checked for consistency. Here, any exterior inter-component constraints can be verified, e.g. concurrency and resource constraints.

Component Configuration

Once components are selected and higher-level structure assembled, this step generates component-specific configuration files for deployment. Parameter values are selected from feasible regions specified by the requirements and matched by the component parameter space.

The system architecture and the main methodology components are described in section 3.

The paper presents in the next section the technologies considered in this project. Section 3 describes the top-down decomposition methodology. Section 5 summarizes the paper with a discussion and conclusions.

2 Background

In this section we examine briefly several technologies that are relevant for OPP and this project and relate to software development and semantic modeling.

2.1 Model Driven Architecture

Software development has struggled for a long time to reach the level of productivity, quality and predictability achieved in other engineering domains. One approach advocated by the Object Management Group (OMG) is the Model Driven Architecture (MDA) [18]. MDA supports model-driven engineering of software systems and provides guidelines for specifying system structures as models. MDA separates the fundamental logic behind a specification – the Platform Independent Models (PIM) – from the specifics of the particular system that implements

it – the Platform Specific Model (PSM).

A complete MDA specification consists of a base PIM, one or more PSMs, plus sets of interface definitions that describe how the base PIM is implemented on a different platforms. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one for each platform selected by the application developer.

The underlying mechanism for defining PIMs is the Meta Object Facility (MOF). The MOF is MDA's foundation specification for modeling languages; MOF compliance allows UML structural and behavioral models to be transmitted via XMI [20], stored in MOF-compliant repositories, and transformed and manipulated by MOF-compliant tools and code generators. The UML is used for building models that comply with the MOF.

Although MDA was developed primarily for distributed systems, its methodology and model technologies are relevant to the OPP project. Maintaining software application frameworks and components during their lifetime on soft/hardware platforms that keep evolving can be a daunting task. MDA PIMs separate the application logic from the platform specific details. This separation simplifies porting an application to a new platform, as only a PSM is required, which is independent from the application.

An introduction to MDA is available online [5]. [15] lists MDA tools and products.

2.2 The Unified Modeling Language

UML [19] was designed by the OMG to assist software developers to specify, visualize, and document models of software systems, including their structure, behavior and designs. UML provides a standardized graphical notation that developers use to build an abstract model of a system, also called the UML model.

The recent revision of the language, UML 2.0, added extensibility to the language introducing a profile mechanism to customize the language. New concepts can be introduced to the language by defining a stereotype. UML version 2.0 defines thirteen types of diagrams, divided into three categories: six diagram types represent static application structure; three represent general types of behavior; and four represent different aspects of interactions:

Structure Diagrams include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.

Behavior Diagrams include the Use Case Diagram (used for requirements gathering); Activity Diagram, and State Machine Diagram.

Interaction Diagrams derived from the more general Behavior Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

UML has been criticized for being difficult to learn, very complex and having imprecise semantics. UML is ineffective when developers must maintain manually synchronize diagrams with code, as this may lead to inconsistencies. Ideally, UML tools should generate all implementation code from UML diagrams.

UML developers may use the declarative **Object Constraint Language** (OCL) to describe rules that apply to UML models. The OCL is a precise text language that provides constraint

and object query expressions on any Meta-Object Facility model or metamodel that cannot otherwise be expressed by diagrammatic notation.

For the OPP project, we assume UML will continue to be the language of choice at Motorola for system architecture design. UML has a wide user base and is supported by many tools, such as I-Logix Rhapsody [12], IBM/Rational Software Architect and Modeler [13], Telelogic System Architect/Developer [24] and Eclipse [7]. A comprehensive list of UML 2.0 tools is available from the OMG at [17].

In this project we will continue to use Rhapsody, as it provides support for UML 2.0, OCL, XMI import/export, SysML and has extensive features for development of real-time applications. We will also consider Eclipse, that has the advantage of being open-source, with publicly available plug-in interfaces. The top-down decomposition methodology will initially work with XMI, and Eclipse would later provide the least cost path to integration with a modeling tool. See section **??** for more details.

2.3 The System Modeling Language

The SysML (Systems Modeling Language) [16] is a domain-specific visual language for systems engineering applications, adopted by the OMG in 07/06. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities.

SysML is defined as a UML 2.0 profile and reuses the notations and semantics of UML 2.0 diagrams. The advantages of this are: (a) SysML can be used together with UML in the same projects to address systems engineering aspects, (b) SysML is extensible using UML's profile and stereotype mechanisms, (c) familiarity for UML users, (d) growing support from the UML community. Unfortunately, SysML also inherits UML's problems with lack of formal semantics, bloat and complex notation.

SysML reuses seven diagrams from UML 2.0, with slightly different semantics, and introduces two new diagrams. We are interested in SysML for its capability to express system-level design concepts and for its support for requirements management with two new diagrams:

Parametric diagram : shows parametric constraints between structural elements. Useful for performance and quantitative analysis.

Requirement diagram : shows system requirements and their relationships with other elements. Useful for requirements engineering.

A SysML parametric diagram developed with Rhapsody is depicted in Figure 3. The diagram describes the constraints and arithmetic relationships defined between system attributes for a mobile phone management application.

SysML has support from I-Logix, Telelogic, IBM, Mentor Graphics, PivotPoint Technology Corporation, Sparx Systems, Vitech Corp and other. Popular tools that integrate SysML are Rhapsody and Telelogic TAU. For more information on the language, please consult the web references [16] and [10].

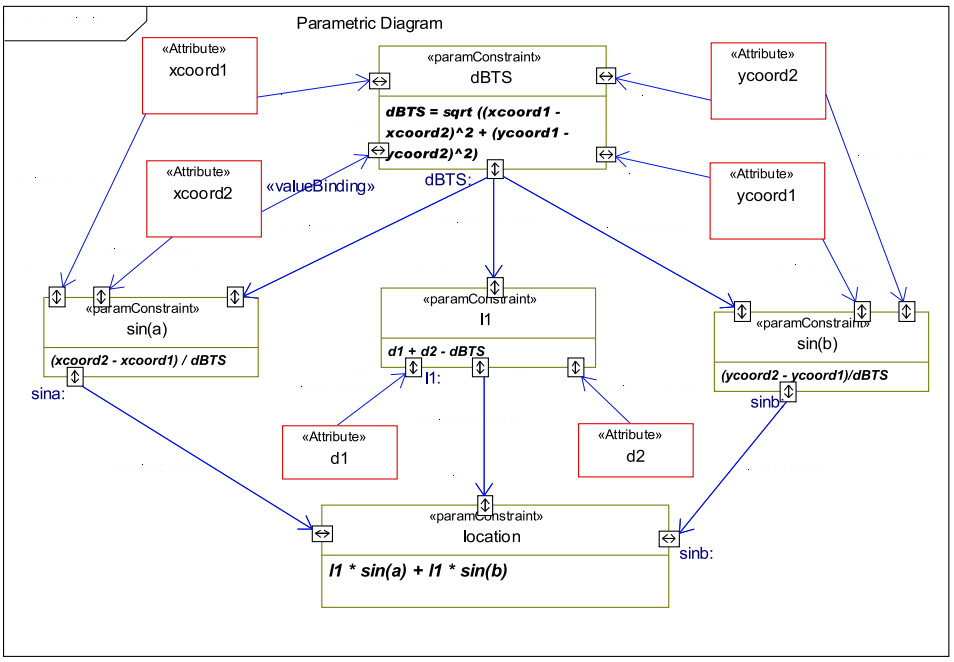

Figure 3: SysML parametric diagram for a cell phone application (from I-Logix Rhapsody).

2.4 Requirements Management

Proper specification and tracking of product requirements is mandatory for successful projects. Common approaches to capture requirements are based on:

text files , where requirements (use cases and functional) are described in natural language and broken down to be easier to manage in tabulated form. Requirements files follow a certain format centered on organization and readability. Marketing Requirements Documents (MRDs) have a text-based file format.

modeling languages such as UML modelers supports several diagrams that can be used to describe systems requirements, such as use case diagrams, activity diagrams and interaction diagrams. SysML introduces special diagram types for requirements specification – the requirement diagram and the parametric diagram. SysML allows model elements to be traced back to individual requirements. Specification of requirements using modeling tools improves team communication.

requirements management tools are multi-user environments used to capture, link, trace, analyze, and manage changes to requirements. These tools are used mostly for distributed development of complex products. Traceability analysis is used to validate specifications and to identify unaddressed requirements. Established RM tools are Telelogic DOORS [23] and IBM/Rational Requisite Pro [21].

For our project, we need to define a semantic bridge between requirements and components. From the methods above only SysML and UML 2.0 manage to partially describe requirements semantics in a formal, machine-readable format, when described with OCL.

As part of our methodology, we develop domain-specific ontologies for describing requirements semantics with the ODL language. UML/SysML requirements models will be used to compile ODL files with specific requirements. Additional semantic information must be captured by the user for aspects not covered by SysML/UML.

Capturing semantics from requirements can be simplified and better automated in case the design team uses UML 2.0 profiles specialized for the problem domain or *Domain Specific Modeling Languages* (DSML). DSMLs formalize the application requirements, architecture, and the behaviors of specific domains (e.g. cell phone applications avionics, grid computing, banking). DSMLs are specified with metamodels that precisely define the semantics of domain concepts, capabilities, concepts and the relationships between concepts. For more about DSMLs consult the documents at the DSM Forum [9].

Domain-specific UML 2.0 profiles or a DSML provide the following benefits: (a) define an unequivocal semantic for domain concepts, (b) reduce bloat and confusion by restricting the domain vocabulary, (c) optimize expressiveness for the target domain, and (d) high degree of code generation and better code quality.

2.5 Semantic Modeling

A key element of our methodology for top-down decomposition is to describe the semantics of requirements and components. Beginning in the late 1990s the World Wide Web Consortium (W3C) began developing knowledge representation and processing mechanisms for the internet. The work led by Tim Berners-Lee on the Semantic Web project [31] has resulted in a series of standardized languages designed for operation in a distributed network, such as internet. These have garnered wide support from the industry and tool developers. The W3C semantic web languages present the advantage (when compared with older knowledge representation techniques) that they can support development of distributed knowledge bases with their capacity of referring to remote resources via Unified Resource Identifiers combined with network transfers.

2.5.1 The Resource Description Framework and the RDF Schema

The Resource Description Framework (RDF) [30] was designed primarily to represent metadata for web resources. RDF is based on XML and was designed on the idea that information on the web can be represented in statements that describe things by the properties they have:

<subject> <property> <object>

For instance, a statement saying that *the Olympus D510 camera has a resolution of 3 megapixels* can be written as an XML triplet with the following RDF code:

```
<?xml version='1.0'?>
  <rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:opp='http://motorola.com/opp/terms/'>
```

```
<rdf:Description rdf:about='http://motorola.com/opp/components#OlympusD510'>
   <opp:hasResolution rdf:datatype='http://motorola.com/opp/units#megapixel'>
   3
   </opp:hasResolution>
<rdf:Description>
</rdf:RDF>
```

The subject is *the Olympus camera*, the property is *has resolution* and the object is *3 megapixels*. The second and the third lines identify the XML namespaces. Resources in RDF are represented with Uniform Resource Identifiers references [22]. The URI part could denote a real location of a web resource, but is not mandatory. Using unique URIs for each concept avoids name clashes. URIs ensure that concepts are not just words in a document but are tied to a unique definition that everyone can find on the Web.

The hasResolution tag describes the property. Multiple statements may share common resources as subjects and objects, forming a graph. We can specify that the Olympus D510 camera has a USB interface by adding the following code:

```
<rdf:Description rdf:about='http://motorola.com/opp/components/#OlympusD510'>
   <opp:hasPCInterface rdf:resource='http://motorola.com/opp/interfaces#USB'/>
<rdf:Description>
```

The beauty of the RDF is that this second statement can be made separate from any prior descriptions of the subject/object/property. It can be loaded into a knowledge base from a resource on the web. Thus, the RDF subject-property-object triplet graph may have concepts distributed on a network, such as the internet, accessible with common web protocols. RDF files referring to common concepts can be defined by different people and can be extended as long as consistency with prior descriptions is maintained.

RDF does not enforce any semantic structure for the statements and this makes processing information encoded in RDF triplets difficult. RDF Schema (RDFS)[29] support the definition of a particular vocabulary that is used for RDF attributes such as hasResolution. RDFS allows the specification of the type of object to which the attributes can be applied. Hence, the RDF Schema approach implements a basic type system for RDF models. This type system uses predefined terms such Class and subClassOf that can be used to describe domain-specific schemata.

The notion of class in RDFS is different from a class in OO Software Engineering, permitting an instance (resource) to belong to more than one class. This is indicated using the type property. An RDFS class is analogue to the mathematical notion of set. Using the RDFS concept of subClassOf, entire class hierarchies can be defined. subPropertyOf does the same for properties. In addition, constraints on properties can be specified using domain and range constructs. With these, RDF Schema can be used to extend both the vocabulary and the intended interpretation of RDF expressions. An RDF Primer is available from the W3C [28].

2.5.2 The Ontology Web Language

The Ontology Web Language (OWL) [27, 3] is used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. Thus, OWL can be used to specify ontologies, as defined in section 1. On the semantic web, the term *ontology* is also used for a document or a file that formally defines the relations between concepts in a specific domain. The most typical kind of ontology for the Web has a taxonomy and a set of inference rules. A taxonomy defines classes of objects and the relations among them.

For instance a *cell phone is a type of a* mobile terminal, and a mobile terminal has a *location*. Relations between entities can be expressed easier if properties are assigned to classes and subclasses can inherit them. We can infere that a *cell phone* also inherits the *location* property. With inference, even more information can be extracted from existing ontologies. For instance, assume a *frequency allocation* called FA requires that, in a specific *location* called L, only a channel C must be used. A reasoning engine would infer that it a cell phone associated with location L must switch to channel C.

RDF is used to build data models between objects (resources) and relationships (properties) between them. This model only support simple semantics. RDF Schema defines vocabularies for describing properties and classes of RDFS objects supporting creation of class and property hierarchies. OWL adds additional vocabulary terms for describing properties and classes, such as relations between classes (e.g. disjointness, union, intersection), cardinality constraints (e.g. ”exactly one”), equality, more types of properties, enhanced characteristics of properties (e.g. symmetric, functional and inverse functional), and enumerated classes.

OWL comes in three variants: (a) OWL Full gives maximum expressiveness and the syntactic freedom of RDF with no computational guarantees, (b) OWL-DL includes all OWL language constructs, with several restrictions: while a class may be a subclass of many classes, a class cannot be an instance of another class), and (c) OWL-Lite supports those users primarily needing a classification hierarchy and simple constraints. For practical reasons OWL-DL provides the best compromise between decidability and expressivity, being supported by many reasoning engines.

In our project we use OWL to define a vocabulary (ontology) of terms from the following domains: cell phone application requirements, system and software components, and a subset of MOF for describing the design model (UML). The vocabulary we propose is called the OPP Design Language (ODL) and describes instances of classes from these domains and their properties. ODL files are also called ontologies.

2.5.3 Reasoning and Semantic Models

Ontology languages allow users to write explicit, formal conceptualizations of domains models. The main requirements for ontology languages are (from [3]):
1. a well-defined syntax
2. a well-defined semantics
3. efficient reasoning support
4. sufficient expressive power
5. convenience of expression.

A formal semantics describes the meaning of the knowledge, which must be unequivocal for both people and machine to make reasoning possible. A reasoner infers new information on class membership, class equivalence and checks consistency of instance class membership. Additional rules can be defined that change the contents of the knowledge base depending on existing instances.

For the Top-down Software Decomposition project reasoning is used to perform the search in the design space based on requirements and components specifications. The reasoning engine will derive new knowledge and answer queries from the main application. Mainly, reasoning can: (a) check consistency of knowledge base (ontologies), (b) validate relations between classes, (c) automatically classify instances into classes.

Reasoning with OWL ontologies has the advantage of integrating knowledge bases (ontologies) from different sources from the web and from different authors, provided new information does not contradict existing ontologies. We now survey several reasoning platforms compatible with OWL, that can be used in the Top-down Software Decomposition project.

Jena, A Semantic Web Framework for Java

Jena [14], developed by the HP Labs Semantic Web Programme, is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS, OWL, SPARQL (a semantic web query language) and includes a rule-based inference engine. Jena is open source and has been used for developing many knowledge processing applications.

The Jena Framework includes: an RDF Java API, reading and writing RDF in RDF/XML, N3 and N-Triples, an OWL API, in-memory and persistent storage, a SPARQL query engine, support for pluggable query engines and building inference engines. Jena provides most features needed for a knowledge processing application on the semantic web. In addition, Jena provides a tool for generating Java classes from OWL/RDFS vocabularies. Jena comes with 4 builtin reasoners, including a generic rule reasoner with backward chaining. Backward chaining is an inference technique that starting from a goal statement can return the necessary facts that support the proof. For the Top-down Software Decomposition project, Jena is used for prototyping the decomposition driver and consistency checking.

Jess, the Rule Engine for Java

Jess [1] is a rule engine and scripting environment written in Java that reasons on knowledge provided in form of declarative rules. The user specifies rules in a text based or an XML-based format and data (facts). The rule engine tries to match rules with the data. Matched rules *fire* executing prescribed actions and possibly changing the knowledge base by adding/remove/changing available facts.

Jess uses a language compatible with the CLIPS system [6] and features a scripting engine that permits calling Java code from rules, and access to the knowledge base from Java code. The drawback of Jess is that it supports OWL only via an external ontology compiler.

Protegé

Protegé [2] is an open source ontology editor and knowledge-base framework developed at Stanford that supports OWL and a custom frame-based ontology format. Protegé is written in Java, has a plugin framework that is extensible, support a Java API and implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and

manipulation of ontologies in various representation formats.

Protegé enables users to:
- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers (Jess).
- Edit OWL individuals for Semantic Web markup.

We will use Protegé in our project to define the ODL vocabulary and write the ODL ontologies (OWL instances and properties). The Jess plugin will help us prototype processing rules and evaluate our approach.

3 The Architecture for Top-down Software Decomposition

This section describes our approach for top-down decomposition . From section 1, we reiterate our project goals to build a methodology that assists designers by matching product requirements with component capabilities to:

- find component structures that satisfy requirements
- track components to requirements
- check requirements consistency
- produce component configurations.

We begin by reviewing the high-level system architecture, shown in Figure 4.

The OPP Design Language

The OPP ontology design language (ODL) is implemented by experts in ontologies working together with experts in the domains of software design (MOF), cell phone application requirements and components.

UML/SysML Requirements

Product requirements are described by product managers and marketing people in a modeling language and translated by our system to ODL ontologies from XMI. Incomplete information from requirements must be entered using a wizard mechanism. From UML/SysML models, the knowledge extraction step applies transformation rules to compile ODL ontologies that can be processed by the reasoner.

UML/SysML Components

Components from libraries are tagged with ODL ontologies (model metadata) that describe semantics not covered by UML/SysML. The remainder of component semantic data is extracted directly from the UML/SysML model diagrams, such as relationships, attributes, interfaces, ports, etc. The component models are stored in XMI format on a filesystem or in a database.

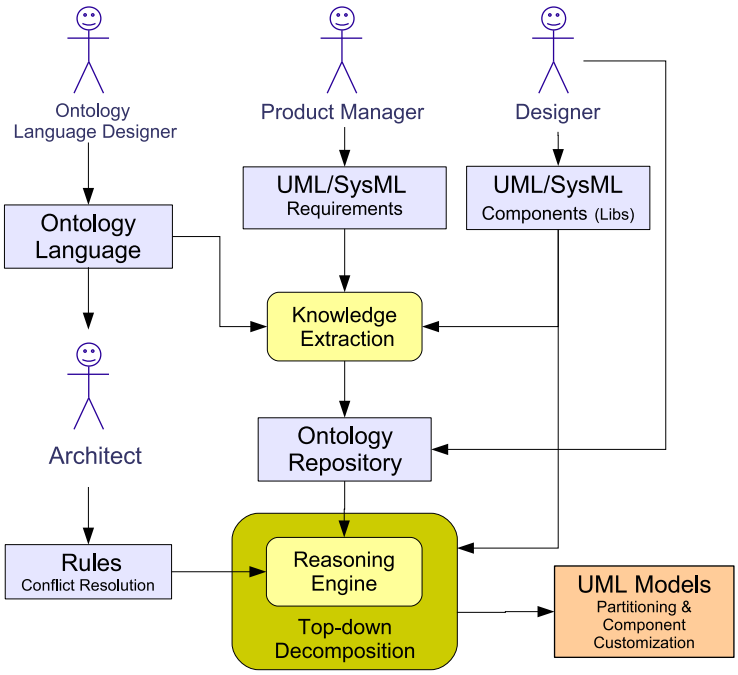

Figure 4: Architecture of the Top-down Software Decomposition methodology.

The Ontology Repository

ODL ontologies encoding requirements and component semantics are stored in their OWL format in the ontology repository (OR). The implementation of the OR depends on the reasoning engine. Jena provides persistence interfaces for both in-memory storage and database storage.

Rules for Conflict Resolution

The system architect must define a set of rules that describe how eventual conflicts are handled if they occur. These policies are written in the rule language supported by the reasoning engine.

Top-down Software Decomposition

This component is the main driver for this application. It embeds a reasoner, formulates queries and interprets results. It then formats the results adequately, in form of status reports and UML models, plus component configuration metadata. The Top-down Software Decomposition methodology eventually will be integrated at the tool level and changes/additions to UML/SysML structural diagrams will be done online. Currently, communication with the modeling tool is via XMI.

3.1 ODL Semantic Domains

This section presents briefly the scope of the OPP Design Language. The ODL must be able to express concepts from the domain of product requirements, components, and the MOF – for supporting UML/SysML concepts.

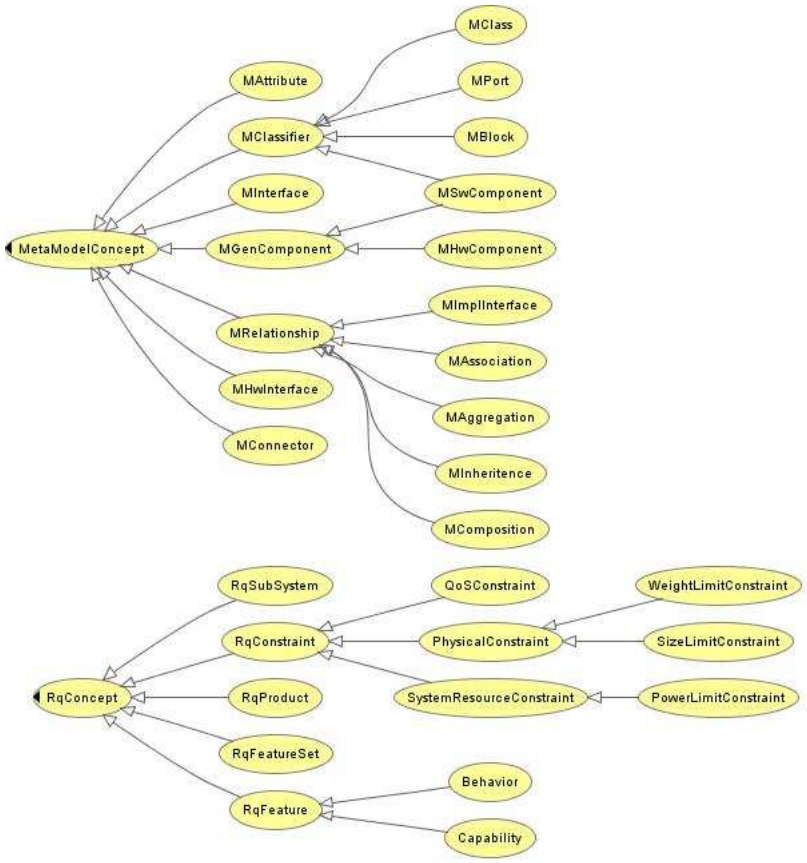

Figure 5: A snapshot of the OWL class hierarchy that forms the ODL metamodel.

A snapshot of the ODL class metamodel is shown in Figure 5.

The top part of the hierarchy supports the MOF 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 hierarchy (derived from class RqConcept) supports requirements specifications. It features sub-system 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 ontologies are developed with Protegé, and can be exported in a variety of formats, besides OWL, compatible with a variety of tools. The ODL metamodel that forms the language to be used for Top-down Software Decomposition , is extensible and can integrate third-party ontologies.

4 Related Work

This section summarizes related work and provides pointers to more information.

Domain-Specific Modeling is an effective mechanism for improving the design productivity considerably, since all concepts and diagrams are optimized for the specific domain. A platform for building domain-specific modeling tools is the Generic Modeling Environment from Vanderbilt [26]. The configuration is accomplished through metamodels specifying the modeling paradigm (modeling language) of the application domain. The modeling paradigm contains all the syntactic, semantic, and presentation information regarding the domain; which concepts will be used to construct models, what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and rules governing the construction of models. The modeling paradigm defines the family of models that can be created using the resultant modeling environment.

The metamodeling language is based on the UML class diagram notation and OCL constraints. The metamodels specifying the modeling paradigm are used to automatically generate the target domain-specific environment. The generated domain-specific environment is then used to build domain models that are stored in a model database or in XML format. These models are used to automatically generate the applications or to synthesize input to different COTS analysis tools.

Another approach for increasing design productivity is through model transformation. A model transformation takes as input a model conforming to a given metamodel and produces as output another model conforming to a given metamodel. OMG has proposed QVT (Queries /Views/Transformations) [4]. QVT defines a standard way to transform source models into target models. There are several ideas in this proposal. One is that the source and target models may conform to arbitrary MOF metamodels. Another one is that the transformation program is considered itself as a model, and as a consequence also conforms to a MOF metamodel.

A good study of use of ontologies in software engineering is given by Hesse in [11]. The author has some interesting conclusions. Ontology development as part of a software project must proceed in parallel and with a wider horizon. Usually, ontologies will address multiple projects and will span several teams. An interesting interaction between ontology and software development is noticed: that system analysis implies the investigation of existing ontologies and the transfer of codified knowledge for the application domain being considered, and system implementation and operational use imply feedback of the project results and experiences to the ontology developers/maintainers and have to be incorporated there in order to keep the ontology alive. In addition, ontologies can be used for transferring knowledge from project to project and to promote communication.

The W3C workgroup note from [25] gives a detailed survey on ontology driven architectures and uses of the semantic web in Systems and Software Engineering. The authors cover especially the area of Knowledge-based Software Engineering. Several interesting ideas are contemplated: (1) use the semantic web as a tool for classification or as a mechanism for “rigorously describing, identifying, discovering and sharing artifacts amongst discrete subsystems, systems and systems’ design teams both during design and at runtime”; (2) use ontologies for formal model specification due to the unambiguous qualities of information representation of the semantic web; (3) software lifecycle support; and (4) use the semantic web as a system for “ for runtime information and artifact sharing“. The last concept is followed by our Top-down

Software Decomposition project. We use semantic web runtime capabilities for manipulating knowledge to improve design quality and reduce costs.

Finally, the work in [8] proposes an architecture for online semantic-based service composition in applications with sensor networks. While this is a very different domain from OPP, the approach for synthesizing higher level services from lower level services provided the technical inspiration for our project.

5 Conclusions

In this whitepaper we summarize the objectives and the technical approach of the Top-down Software Decomposition project and we describe a methodology for improving the design and for reducing the costs of the design process. 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 both domains.

We also survey the main technologies involved in this project. The Model Driven Architecture provides the backdrop for a formal software development process and for visual modeling tools. We describe the UML and SysML modeling tools that are commonly used in the industry for requirements analysis and for software design. The main semantic technologies from the semantic web are also covered: RDF, RDFS, OWL and several reasoning engines.

References

[1] Jess, the rule engine for Java. http://www.jessrules.com.

[2] Protége: ontology editor and knowledge-base framework. protege.stanford.edu.

[3] G. Antoniou and F. van Harmelen. *Handbook on Ontologies in Information SystemsHandbook on Ontologies in Information Systems*, chapter Web Ontology Language: OWL. Springer-Verlag, 2003. http://jeogoodjob.250free.com/data/OntoHandbook03OWL.pdf.

[4] M Bohlen. Qvt and multi metamodel transformation in mda. http://galaxy.andromda.org/jira/secure/attachment/10780/QVT+article+mbohlen+2006.pdf.

[5] Alan Brown. An introduction to Model Driven Architecture. http://www-128.ibm.com/developerworks/rational/library/3100.html.

[6] CLIPS. A tool for building expert systems. http://www.ghg.net/clips/CLIPS.html.

[7] Eclipse. EMF-based UML 2.x Metamodel Implementation. http://www.eclipse.org/uml2/.

[8] Kamin Whitehouse et al. Semantic streams: a framework for declarative queries and automatic data interpretation. Technical report, Microsoft Research, 2005. ftp://ftp.research.microsoft.com/pub/tr/TR-2005-45.pdf.

[9] The Domain-Specific Modeling Forum. http://www.dsmforum.org/.

[10] The SysML Forum. http://www.sysmlforum.com/.

[11] Wolfgang Hesse. Ontologies in the software engineering process. In R. Lenz et al., editor, *EAI 2005 - Tagungsband Workshop on Enterprise Application Integration*, 2005.

[12] I-Logix/Telelogic. The Rhapsody UML Modeling Tool. http://www.ilogix.com/sublevel.aspx?id=53.

[13] IBM/Rational. Rational Software Architect. http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html.

[14] Jena. A semantic web framework for java. http://jena.sourceforge.net.

[15] OMG. MDA committed companies and their products. http://www.omg.org/mda/committed-products.htm.

[16] OMG. Omg systems modeling language. http://www.omgsysml.org/.

[17] OMG. OMG's list of UML 2.0 Tools. http://www.uml.org/.

[18] OMG. The Model Driven Architecture. http://www.omg.org/mda.

[19] OMG. The Unified Modeling Language. http://www.uml.org.

[20] OMG. XML Metadata Interchange (XMI). http://www.omg.org/technology/documents/formal/xmi.htm.

[21] IBM Rational. RequisitePro. http://www-306.ibm.com/software/awdtools/reqpro/.

[22] Berners-Lee T., Fielding R., and Masinter L. IETF RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax. http://www.isi.edu/in-notes/rfc2396.txt, 1998.

[23] Telelogic. DOORS: Solution for Requirements Management. http://www.telelogic.com/corp/products/doors/index.cfm.

[24] Telelogic. Telelogic SYSTEM ARCHITECT for Enterprise Architecture. http://www.telelogic.com/corp/products/tau/.

[25] Phil Tetlow, Jeff Pan, Daniel Oberle, Evan Wallace, Michael Uschold, and Elisa Kendall. Ontology driven architectures and potential uses of the semantic web in systems and software engineering. http://www.w3.org/2001/sw/BestPractices/SE/ODA, 2006.

[26] Vanderbilt University. The generic modeling environment. http://www.isis.vanderbilt.edu/projects/gme/.

[27] W3C. The ontology web language. http://www.w3.org/2004/OWL.

[28] W3C. The RDF Primer. http://www.w3.org/TR/rdf-primer/.

[29] W3C. Rdf vocabulary description language 1.0: Rdf schemardf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/.

[30] W3C. Resource Description Framework. http://www.w3.org/RDF/.

[31] W3C. The semantic web page. http://www.w3.org/2001/sw.

Addendum

Languages Used	References	Comments
Java	sun.com	Java Language
UML 2.0	[19]	standard design language
SysML	[16]	Emerging standard for systems engineering. Requirements diagram
XMI	[20]	MOF portable format
OWL	[27, 3]	The Ontology Web Language

Tools	References	Comments
Protége	[2]	Semantic web framework (Java); build ODL and ontologies
Jess	[1]	A rule engine for Java; rule-based reasoning
Jena	[14]	A semantic web framework for Java; supports OWL processing and reasoning
Rhapsody	[12]	UML modeling tool for real-time and embedded systems; has XMI support and read-only model access
Eclipse	[**?**]	EMF-based UML 2.x modeling tool; open-source; full access to internal UML 2.0 models; extensible plugin architecture

Promoting Reuse, automation and productivity enhancement

The Top-down Software Decomposition methodology aims to automate parts of the initial design phases. We develop an ontology language that is used to describe the semantics of product requirements and components. Ontologies with such descriptions are processed by an automatic reasoner and, as result, requirements are checked for consistency, components are automatically selected s.t. constraints and requirements are satisfied and partial UML design models are generated with new configurations.

This project

- promotes component reuse: by tagging components with metadata describing capabilities and constraints, these can stored in libraries and managed easier,
- increases requirements quality: by checking consistency of requierements and finding compliant components,
- automates the design process: by assisting the designers with the search for the proper components and configurations based on requirements and constraints.

Innovative Ideas

The concept of ontology-based UML architecture synthesis from requirements and component

specifications may be original, although the related concept of bottom-up semantic service composition has been applied in sensor networks and for web services. The underlying mechanism is backward-chaining, a technique that has beed used for theorem proving.

The OPP vision for a 24 day design cycle requires integrating the top-down decomposition methodology with the design toolset. Eventually, the technologies described in this whitepaper will have to be integrated with requirements management and MDA PIM design with UML/SysML.

Issues

Major issues are related to the size of the design space: (1) the complexity of the MOF and UML metaschemata, (2) the huge scope of the problem domain, (3) the big variety of sw/hw components. Defining ontologies to cover all these aspects is a daunting task by itself. This is beyond the scope of this project. To mitigate the design space problem we will focus on proving the top-down decomposition methodology on a small application. With enough effort and by collaborating with standardization bodies, component creators and modeling tool producers it is definitely possible to build a streamlined top-down decomposition design flow into the OPP design process, generic enough and extensible to accomodate existing and upcoming applications with small effort.

Another issue is requirements specification. Current approached for requirements specifications do not address application semantics in a formal fashion. The ODL language we propose would permit specification of product functional requirements and QoS constraints. However, it is not feasible to ask non-technical people to design specs in the highly-specialized ODL OWL-based language. A solution is to develop one of: (a) a domain-specific requirements (visual) modeling tool that generates ODL from user-specified models, (b) mechanisms for compiling ODL ontologies from requirements expressed in UML/SysML models. The second approach may still need editing ODL for semantics not covered by UML/SysML.

Project Roadmap

This is the current schedule, as of July 2006. It may change following August discussions with Motorola and task re-prioritization.

1. ODL ontology language prototype for a subset of UML and narrow application/component domain: Sept. 2006
2. ODL compilation from component UML models : Nov. 2006
3. requirements checking, component selection: Mar. 2007
4. UML structure diagram synthesis, model configurations: June 2007
5. methodology demo for a location-based services application: Fall 2007