Model-Assisted Software Development: Using a 'semantic bus' to automate steps in the software development process

Editors: Krzysztof Janowicz, Pennsylvania State University, USA, and Pascal Hitzler, Wright State University, USA.
Solicited Reviews: Axel Polleres, DERI, National University of Ireland, Ireland and Krzysztof Janowicz, Pennsylvania State University, USA.

Kunal Verma*, Alex Kass
Accenture Technology Labs, 50 W. San Fernando Street, Suite 1200, San Jose, CA, USA

Abstract. The early phases of the software-development lifecycle (SDLC) for enterprise-scale systems - in particular, requirements elicitation, functional design, and technical design – are difficult to automate because they involve the application of several different kinds of domain knowledge. In this paper, we will provide a vision of how creating semantic models of domain knowledge used in each phase, and defining semantic representation, through which tools in the various phases can communicate knowledge across phases, can help provide more automation both *within* and *across* these phases. We refer to the collection of semantic models needed to support this automation as *the semantic bus for software development*. We refer to the semi-automated process that we envision making use of this bus to support the SDLC as, *Model-Assisted Software Development (MASD)*, which is a variation on the Model-Driven Development idea. We will describe tooling we have built which realizes part of this vision, and will outline a roadmap of potential research opportunities in this space.

Keywords: Software Engineering, Semantic Bus, Model-Assisted Software Development

1. Introduction

The effective execution of each phase of a complex process often involves applying a type of knowledge that is specific to that phase of the process, as well as accessing and building on the knowledge created in other phases. The effective automation of complex processes involving multiple teams of people generally requires a loosely coupled set of tools supporting each phase of the process. Keys to success include supporting the application of phase-specific knowledge by each tool and communication of knowledge *across* phases and tools.

The early phases of enterprise-scale software development – from specification through functional and technical design - represent an important and complicated example of this challenge. These activities typically involve experts in multiple business domains, as well as expert functional and technical architects. Each group of experts contributes their respective types of knowledge to the process: For instance, in the elicitation sessions within the requirements phase, expert stakeholders' knowledge of business objectives and processes is leveraged to identify key requirements that the solution must fulfill. During functional design, architects leverage their knowledge of functional and software frameworks to create functional design artifacts. In technical design, another set of architects leverages their knowledge of architectural patterns to choose relevant technical services and create technical design artifacts.

The complex, knowledge driven nature of the work in the early phases of the software-development lifecycle makes semantic technologies very relevant to their automation– both to represent the knowledge that gets applied in each phase, and to support knowledge transfer between phases. However, there are

* Corresponding Author. E-mail: k.verma@accenture.com

no widely-deployed commercial tools that support the use of semantic models for the specification and design of enterprise systems. We believe that this is one reason that these up-front phases remain the least automated portions of the software-development lifecycle.

Filling this gap is important because it could help address the well-documented problems associated with these early software-engineering phases: Defects in requirements and design are common and expensive, often leading to extensive rework or delays in software deployment [17]. Defects can arise from failure to leverage applicable knowledge *within a phase*, or from a failure to reason *across phases* to keep the deliverables of various phases in synch with each other; both of these issues could be mitigated by tools that support automated application of appropriate knowledge models to the development and review of each activity's deliverables.

In this vision statement, we will describe how semantic web technologies, combined with intelligent tools that leverage knowledge encapsulated in the semantic models, can be used to support the early phases of the SDLC. We refer to the collection of semantic models needed to support this automation as *the semantic bus for software development*. We refer to the semi-automated process that we envision making use of this bus to support the SDLC as *Model-Assisted Software Development (MASD).* As in the well-known vision often referred to as *Model-Driven Development* (MDD), the MASD process we envision relies on models that can be transformed to support progression between phases. However, as we shall describe, the role of the models, and the nature of the automation we envision is rather different than in traditional MDD.

Of course, our vision has not yet been fully realized in software. However, to make our discussion as concrete as possible, we will briefly describe some tooling which has been built to support aspects of the requirements and design activities, which begins to realize parts of the vision. Our hope is that over time, not only will we be able to implement more of the pieces, but that a rich eco-system of tools from other developers will emerge to support each phase of the SDLC, all of which can be loosely coupled through the envisioned semantic bus, allowing development teams to mix and match freely to suit their specific needs.

2. Sketching the Semantic Bus

We see a need for at least two kinds of semantic models, or ontologies, making up the semantic bus.

1. Artifact inter-communication models: A set of linked models for representing the artifacts produced by the various phases of software engineering. These models are intended to be independent of any business domain or underlying technology domain. At the schema level, these models contain descriptions of the component structure of a phase's artifacts, and the relationships between components in one phase and corresponding components in other phases. At the instance level, they contain domain-specific data from the actual artifacts. This part of the bus will be used to transfer data across phases – for instance from requirements to functional design. Some work in this space has already begun in the context of the Open Services for LifeCycle Collaboration (OSLC) [7], where an IBM-led community is creating RDF based representations for artifacts in different stages of software lifecycle such as requirements and testing. The goal of this initiative is not automated generation of artifacts from one stage to the other, rather it is maintaining traceability links across tools. However, having a standardized ontology (or set of linked ontologies) for representing artifacts in different stages of the software lifecycle will be an important enabler of the semantic bus.

2. Domain-specific phase-enablement models: These ontologies model facts about specific business and technology domains relevant to the systems being developed. They will be used by a set of tools for tasks requiring domain specific reasoning, such as gap analysis (during requirements phase) or selection of relevant technical services (during design). One example would be an ontology that describes the typical decision points in the Apache Axis Framework [1]. Another would be an ontology that provides a model of commonly used requirements and capabilities in the banking domain.

3. Using the Semantic Bus in Model Assisted Software Development

In this paper we are focused on three specific activities that come early in the software-development lifecycle: 1) requirements development, 2) functional design, and 3) technical design.

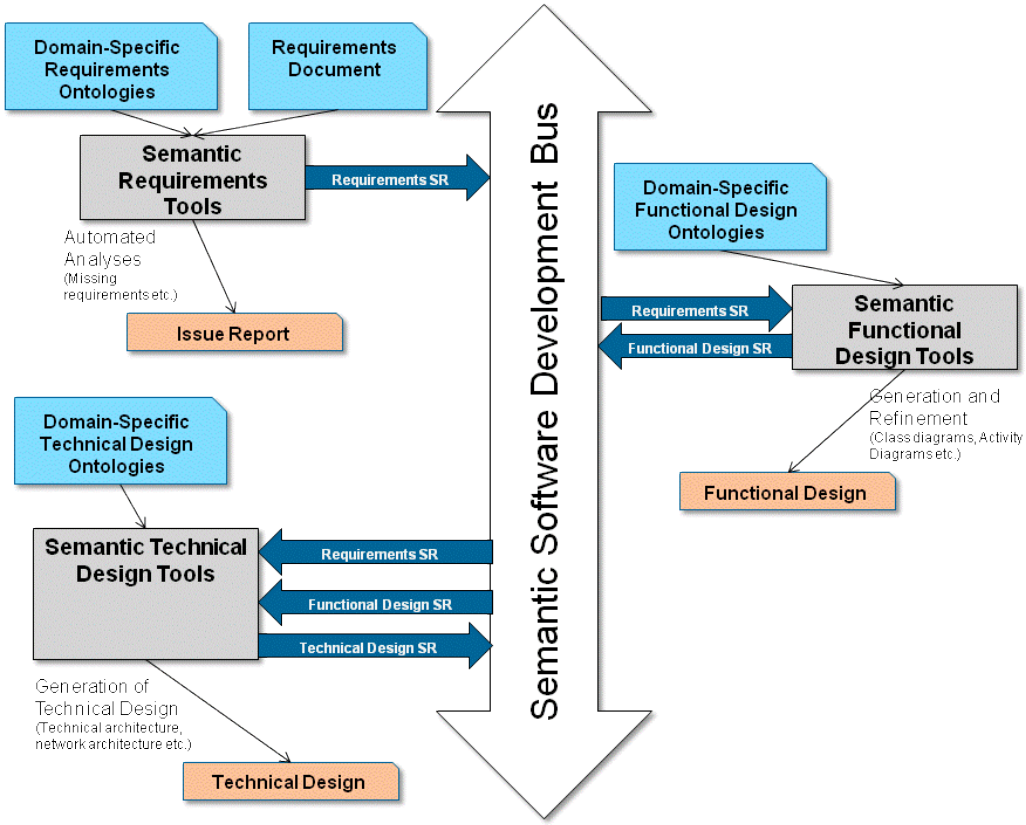

Figure 1: Semantic Bus for Software Development

Figure 1 illustrates the flow of information from the requirements document to functional and technical design. As the figure suggests, we expect all the tools to leverage semantic models in two ways – 1) use the domain specific ontologies to help users perform the specific task and 2) leverage the semantic bus for software development to get data input from previous stages or make their data output available to other tools.

One may ask how our vision is different from the much-discussed vision of Model Driven Development (MDD) [3] or its popular instantiation by OMG-Model Driven Architecture [8]. MDD, where downstream artifacts are automatically generated from models, has been a long-term aspiration of the software engineering community. We obviously agree that models will play an important role in the future of software development. The question really is *what* role? How will models be used in a more automated software development process, and what will that process look like? On some of these questions our vision is a bit different from the most common existing approaches to MDD.

For example, a common activity pattern in the MDD vision is as follows:

- **Step 1:** A human, skilled in one space (such as requirements) uses MDD tool to create a formal model of the deliverable in that space.
- **Step 2:** A system automatically performs a transform on parts of that model to generate artifacts in the downstream space (such as design).
- **Step 3:** Another human, expert in the downstream space, then modifies the automatically-generated artifact.

One characteristic of the MDD vision is that the artifacts created at each stage have to be more formal in order to enable automated transformation. An advantage, at least in theory, is that the transformation from the upstream space to the downstream space (in Step 2) is fully automated. However, as others have noted (for e.g., [2] and [3]), there are some significant challenges which make this conception of MDD very difficult to apply in practice.

1. The human-compatibility problem: Experience has shown that practitioners in an area such as requirements often find thinking in terms of a formal

modeling notation very unnatural. Business Analysts, for example, are used to writing sentences, not creating models. Furthermore, the stakeholders who must sign off on these requirements are used to reading sentences, not model diagrams. Furthermore, while more technical practitioners may not find working with small models challenging, even for them it is very challenging to work with very large models at the scale that will be required for enterprise systems.

2. **The knowledge representation problem:** the amount of knowledge needed to perform step 2 well on real-world artifacts is much greater than what any existing system has. As a result, the amount of modification required in step 3 is often fairly high.
3. **The update problem:** MDD looks good for initial generation of artifacts. However, unless Step 3 is eliminated, the challenge of how to handle updates to upstream artifacts (such as requirements) arises: An architect who spends time understanding the output of Step 2, and then modifies it, will need to repeat that modification step every time the output of step 1 changes even slightly, since the changes made in step 3 will be erased when Step 2 is run again. The effort of having to repeat step 3 for small changes in Step 1 can wipe out the gains from automating Step 2.

To address these concerns, we envision a variation on the MDD theme, which we call Model-Assisted Software Development. One key distinguishing feature of MASD is that the artifacts that the human participants are asked to create and understand are the more traditional human-readable descriptions. In our proposed approach, the formal models designed for automated consumption are automatically created from the human-created documents. For instance, a business analyst is asked to create a well-structured, *natural language* requirements specification. The structured model of the requirements is *automatically* generated by a system capable of analyzing the requirements text. The model then lives alongside the human-generated document, and is automatically kept in synch. The text document is used by human analysts and stakeholders, while the model is available to support automated reasoning, and generation of downstream transformations. So MASD sidesteps the human-compatibility problem by relegating the model to a behind-the-scenes role: in MASD, models are treated as an internal representation for systems to manipulate as much as possible, and for practitioners as little as possible. A second distinguishing characteristic is that we envision more of a *semi*-automated transformation process resulting in higher-quality downstream artifacts. By employing more sophisticated knowledge models in the systems performing activities like Step 2 above, *and* involving humans (with their much larger knowledge models) we seek to minimize Step 3. Human-supplied knowledge helps us side-step the knowledge-representation problem, and minimizing step 3 reduces that pain of the update problem. In other words, one way to think about MASD vs. MDD is this: The MASD vision retreats from the theoretical ideal of complete automation embraced by MDD in favor of a more modest level of automation which can actually be achieved even with the complex artifacts required for real-world enterprise software development.

4. Semantics in Requirements Engineering

Requirements Engineering is the first phase of most software projects. This phase involves business analysts eliciting requirements from stakeholders and documenting them. The business analysts use their domain-specific knowledge to ask relevant questions and guide discussions. Once the requirements are documented, the practitioners use their knowledge to detect issues in requirements documents, such as conflicting or missing requirements. Practitioners also often need to manually perform impact analyses to determine the cost of changes requested by the stakeholders during the course of the projects.

Much of the work in this field (for e.g., [15]) makes the assumption that users will create formal models, instead of natural language requirements specifications, which are still the norm. We believe that the focus should be on generating formal models from natural language text, since it is likely to be the mode of writing requirements specifications for years to come. We have developed a tool called the Requirements Analysis Tool (RAT) [16] that converts a textual requirements document into a semantic RDF graph, which can be queried and reasoned upon. Currently, RAT helps users detect missing non-functional requirements with the help of non-functional requirements ontology. In addition, it automatically generates interaction diagram with the help of some rules. However a number of issues such as conflict detection and impact analysis still remain open issues.

There has been some work on trying to detect missing requirements with the help of domain models. Kaeya and Saiki [1] proposed manually mapping requirements to elements in a domain-specific ontology and they have a measure of completeness based on the number of ontological elements that do not

have any requirements mapped to them. We have recently developed a tool called the Process Model Requirements Gap Analyzer (ProcGap) [17] that uses natural language processing technologies automatically maps requirements to process models, such as "*Order to Cash*," and helps users see the gaps and similarities. ProcGap, for instance, will flag any elements of the standard process model that do not seem to be covered by any project requirement, since these may represent missing requirements.

The early work described above, on leveraging semantic models in requirements analysis, gives some hint of semantic models can achieve in the requirements space, but we believe that there are a number of very important unsolved problems in this area. There is much work to be done in developing domain-specific ontologies. Currently, most software providers provide specifications of their software in textual form. The ability to extract formal specifications from textual specifications will be extremely valuable. In addition, research is needed on what type of reasoning is suitable for various kinds of analyses needed in requirements engineering, such as conflict detection and impact analysis. Finally, we have done some initial work on creating a requirements ontology that can be used for creating downstream artifacts, but much more work needs to be done creating a comprehensive requirements ontology that could serve as the basis for the semantic bus for software engineering.

5. Semantics in Functional Design

The next stage after requirements analysis is functional design. This is another knowledge-intensive stage, in which the functional architect must create a functional design based on their knowledge of software frameworks/tools and their understanding of the requirements. Usually, the functional design is represented as a number of artifacts, ranging from informal figures to more formal UML class or activity diagrams.

We believe that there are two main issues in this phase. First, while there is some tool support in this phase (for e.g., Rational Software Architect helps users generate UML diagrams), there is not much support for the knowledge-intensive decisions that need to be made by the functional architects. For example, if a functional architect realizes that a solution must be service oriented, she must make decisions on which framework to use (for e.g. Apache Axis) and once the framework is decided she must make decisions on the granularity of services and which parts of the framework to be used. Domain-specific ontologies, along with reasoning engines, should be able to assist functional architects with these kinds of decision-making. Second, some knowledge is often lost in the transition between the requirements and functional design phases. This can have various causes, ranging from lack of time, to misinterpretation of some requirements by the functional architect.

A number of researchers have proposed using natural language processing (NLP) to automatically generate design artifacts out of requirement documents and use case specifications. Examples include UCDA [6], LIDA [9], work by Ilieva et al. [4] and Gelhausen and Tichy [12]. We also explored an early prototype called Functional Design Creation Tool (FDCT) for generating a first cut at some functional-design artifacts from requirements based on heuristics [11].

However, none of these approaches (including ours) leverage domain-specific ontologies to generate the design. While these approaches are able to provide a model based on the requirements, they are typically not sufficient to model non-trivial systems, since they do not capture relationships of generated model of existing software libraries or systems. For example, here are some questions that these approaches do not answer:

1. Which modules of SAP do the generated classes interact with?
2. Which classes of frameworks such as Spring or Apache Axis should be should be used to implement some of the generated classes?

We believe that there is a clear opportunity to help functional architects generate functional designs with the help of domain specific ontologies and associated reasoning engines. As with requirements, there is some previous work in this space, but there are a number of open issues and unanswered questions.

6. Semantics in Technical Design

Technical design involves a number of activities, such as deciding the type of architecture (for example, three tier architecture vs. cloud-based architecture) and the types of infrastructural services that are needed such as encryption and logging. Also activities such as choosing appropriate hardware based on the architectural decisions, infrastructural services needed and performance criteria specified using non-functional requirements. It is performed on the basis

of inputs from requirements and functional design. The technical architects also use their domain-specific knowledge to come up with such architectures.

Though this is a very important area of the project, there is practically no work on building knowledge-based tools to support technical design. In a preliminary work [10], we explored how description logics and subsumption-based reasoning can help users select relevant services and hardware based on vendor recommendation. This is an extremely rich area in which to explore the use of semantic technologies.

7. A Deeper Look at the Semantic Bus with the help of an end-to-end scenario

In this section, we will discuss an end-to-end example to illustrate how the semantic bus can be used to transfer information from one phase of software engineering to another. Consider the following steps from a use case:

UC-4-1: Project Manager navigates to employee page in PRMS.

UC-4-2: Project Manager searches for employee record by specifying employee id / employee full name.

UC-4-3: Project Manager modifies the employee record.

UC-4-4: PRMS sends updated employee record to the Employee Repository.

UC-4-5: PRMS sends notification of change to the Resource Manager.

Term	Explanation	Type	Part of
Active projects	Projects that are active	PassiveEntity	
Administrator	Admin of project resource management system.	Person	
Assign Resource Module	Module of PRMS that provides capability for resource assignment.	System	Project Resource Management System
Backup Master Employee Repository	Backup for Master Employee Repository.	System	
budget	Budget for a project	DataAttribute	
Client	Client for the selected project.	Person	
contractor	External contractors who are available for staffing.	Person	
Credit Check System	Vendor system to check credit of employees.	System	
Credit verification report	Report from credit verifier	PassiveEntity	
Deployment Resource Manager	Deployment resource manager of project.	Person	
Employee	Employee of Accenture.	Person	
employee details	Employment and personal details of employee.	PassiveEntity	
employee full name	Full name of employee	DataAttribute	employee record
employee id	Unique ID for the employee	DataAttribute	employee record
employee page	Employee web page	System	Project Resource

Figure 2: Entity Glossary in the Requirements Analysis Tool

The business analyst captures the use case in restricted natural language supported by the requirements analysis tool. In addition, the business analyst creates a glossary that defines the different terms used in the requirements and use cases. The glossary (shown in Figure 2) captures different types of information about the entities, such as the type of the entity (for e.g., person, system, data attribute, etc.) and whether an entity is part of another entity (for e.g., Employee ID is an attribute for Employee record).

The requirements analysis tool then uses a combination of lexical and semantic techniques to analyze the use case text, extract structured content for each use case step and populate the core requirements ontology defined in [16] to create a semantic RDF graph. Thus, the information is transformed from being simple text to semantic graph that can reasoned upon and transformed to create different kinds of reports and models. We show an interaction diagram generated by the requirements analysis tool in Figure 3. This diagram is generated by applying heuristics to figure out which requirements/use case steps represent interactions between the systems and users.

A number of other tools such as UCDA [6], LIDA [9] and FDCT [11] can also be used to generate downstream artifacts such as class diagrams. In addition tools such as ProcGap [17] can be used to compare how the requirements and use cases compare to reference processes and capabilities in that domain. However, none of these tools provide any support using domain-knowledge to create artifacts. While there are tools such as Skyway Builder [12], which can generate Java code based on a model of the Spring framework, there are no such tools for earlier phases of software engineering.

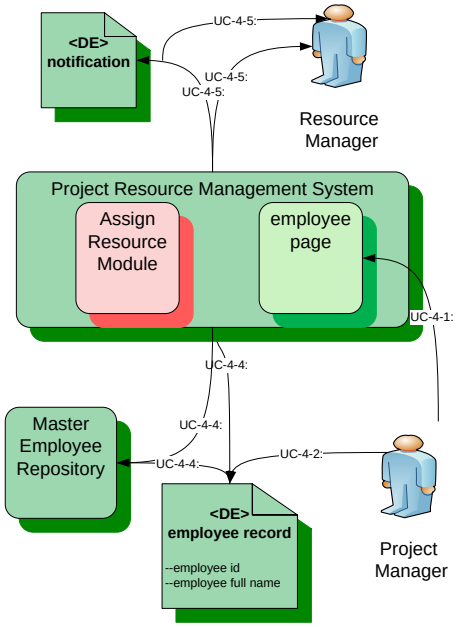

Figure 3: Automatically Generated visualization of use case text using Requirements Analysis Tool

To illustrate our point about tools that leverage domain-specific ontologies, let us assume the above

use case had to be implemented as a Web service using the Apache Axis Framework at the front end and using a batch-style architecture in the back-end. A tool that would guide users in including relevant classes using a semantic model of the Apache Axis Framework would be very helpful. Similarly, a tool that helps users select relevant infrastructure services for the batch-style architecture would also be helpful. Currently, the aspects of functional and technical design, that involve domain knowledge, are done manually with no automated support.

8. Conclusions

In this paper, we have presented a vision for a suite of tools to enable the early phases of the software engineering process. Even though these phases are extremely important, and highly knowledge intensive, there is very little knowledge-based tool support for practitioners. One reason for is that the software-engineering community has tended to focus on tool support for the most tangible phases, such as coding, rather than the early phases which involve messy, less structured artifacts, and require significant amounts of domain knowledge.

This is not the first paper to talk about automating aspects of software development. Interested readers can read a nice summary article [14] that discussed a number of previous vision papers. We believe that there are a number factors that have recently emerged to enable our vision of the semantic bus – 1) the field of natural language processing has evolved significantly enough that converting natural language text to formal models is getting more feasible; 2) OSLC) [7], which uses RDF to represent software-engineering artifacts is gathering momentum and is currently supported by a number of commercial tools; and 3) the Requirements Analysis Tool, which has been deployed at over 400 projects within our organization, has been used by many of those projects to generate high-level design from natural-language requirements.

These are encouraging signs, but, much work still needs to be done, especially around tooling that leverages domain and phase-specific ontologies. We believe that this represents a rich area in which Semantic Web researchers can leverage their skills to build tools that will have a large impact on the state of the art in software engineering.

Acknowledgements

We would also like to acknowledge our collaborators Rey Vasquez, Santonu Sarkar, Vibhu Sharma and Edy Liongosari, who helped us in shaping this vision paper.

References

[1] Apache Axis Framework, http://ws.apache.org/axis/

[2] A. E. Bell, ''Death by UML Fever,'' ACM Queue Magazine 2, No. 1, March 2004.

[3] B. Hailpern and P. Tarr, Model-driven development, The good, the bad and the ugly, IBM Systems Journal, 45(3), 2006.

[4] M. Ilieva and O. Ormandjieva, "Automatic transition of natural language software requirements specification into formal presentation," in Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 392–397.

[5] H. Kaiya and M. Saeki., Using domain ontology as domain knowledge for requirements elicitation. In Proc. Of the IEEE International Requirements Engineering Conference (RE), pp. 186–195, 2006.

[6] D. Liu, K. Subramaniam, A. Eberlein, and B. H. Far, "Natural language requirements analysis and class model generationusing UCDA," in Lecture Notes in Computer Science. Springer-Verlag, pp. 295–304, 2004.

[7] Open Services for Lifecycle Collaboration (OSLC), http://open-services.net/html/Home.html

[8] Object Management Group, Model Driven Architecture, http://www.omg.org/mda/

[9] S. P. Overmyer, B. Lavoie, and O. Rambow, "Conceptual-modeling through linguistic analysis using LIDA," in Proceedings of the 23rd International Conference on Software Engineering, 2001, pp. 401–410.

[10] S. Sarkar and K. Verma, Accelerating technical design of business applications: a knowledge-based approach. In Proceedings of the 3rd India Software Engineering Conference. (ISEC) 2010.

[11] V.S. Sharma, S. Sarkar, K. Verma, A. Panayappan, and A. Kass, Extracting High-Level Functional Design from Software Requirements, 17th Asia-Pacific Software Engineering Conference, 2010.

[12] Skyway Software, http://www.skywaysoftware.com/

[13] Tom Gelhausen, Walter F. Tichy: Thematic Role Based Generation of UML Models from Real World Requirements. IEEE International Conference on Semantic Computing (ICSC), pp. 282-289, 2007.

[14] Michael Uschold, Ontology-Driven Information Systems: Past, Present and Future. Proceeding on the 5th Conference on Formal Ontology in Information Systems (FOIS), pp 3-18, 2008.

[15] A. van Lamsweerde, E. Letier, and R. Darimont, Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transactions on Software Engineering 24(11), 1998.

[16] K. Verma, and A. Kass, Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents, 7th International Semantic Web Conference, 2008.

[17] K. Verma, A. Kass, and R. Vasquez, Aligning Requirements Documents to Industry-Specific Process Models, Technical report, 2010.

[18] K. Wiegers 2003, Software Requirements, Microsoft Press