Ontology-Enabled Traceability Mechanisms

Mark Austin Institute for Systems Research, University of Maryland, College Park, MD 20742. E-mail: austin@isr.umd.edu	Cari E. Wojcik, Civil Security and Response Programs, Raytheon Integrated Defense Systems, Portsmouth, RI 02871. E-mail: cari_e_wojcik@raytheon.com

Abstract. This paper describes new models and mechanisms for ontology-enabled traceability where design concepts are inserted between the already connected requirements and engineering objects Looking ahead, we envision that ontology-enabled traceability will play an important role in the development of network-enabled platforms for analysis, design, and early validation and verification of information-age engineering systems.

Problem Statement

The post-event analysis of recent engineering system failures indicates that, often, the underlying cause of catastrophic and expensive failures is minor mistakes or omission in communication of design intent (e.g., errors in the use of engineering units; errors in the placement of electronic devices on a drawing; logic errors in the implementation of software). The importance of this problem stems from the wide array of engineering applications that have failed in this way. Examples include spacecraft, automated baggage handling systems at airports, and networked services in modern building environments (Jackson 2006; Jones 2004; Sawyer 1999).

The difficulty in finding a good solution to this problem is complicated by industrial-age systems being replaced by information-age systems. As pointed out by Whitney (Whitney 1996), industrial-age systems tend to be dominated by hardware and continuous behavior that can be described by differential equations. Designers can use safety factors to deal with uncertainties in system properties, behavior and design. Information-age systems on the other hand tend to be dominated by combinations of hardware, software and communications, which together are required to provide new types of time-critical services, superior levels of performance, and work correctly with no errors. There are a number of reasons why satisfaction of these goals can be particularly difficult. First, when new technologies are weaved together to achieve new types of functionality, systems can fail in new and unprecedented ways. In the late 1990s, for example, NASA certainly did not anticipate that a miscommunication of engineering units would lead to one of their spacecraft crashing into the surface of Mars. Second, correct functionality for software and communications systems is defined by logic (not differential equations). Not only does the concept of safety factors not apply, as observed in a number of engineering system failures, a small fault in the software implementation can trigger catastrophic system level failures. While it is tempting to assume these errors are caused by bugs in the software, recent studies (Jackson 2006) indicate that almost all grave software problems can be traced back to conceptual mistakes made before the programming even started.

Systems Engineering Community Response. In an effort to improve the accuracy and effectiveness of communication among engineers in the development of real-time systems, the systems engineering community has developed SysML, the Unified Modeling Language (UML) extended for Systems Engineers (Sysml 2003, UML 2003). UML has already found great success in the software engineering community. By introducing a variety of new diagram types to SysML, the hope is that similar success will occur in systems engineering. We believe, however, the name Unified Modeling Language promises more than it can ever deliver. While modest extensions to UML will be useful for documentation, informal analysis, and communication of ideas among systems engineers, both UML and SysML lack the syntax/semantics needed for rigorous analysis and formal verification of system compliance associated with temporal and spatial analysis of physical systems. While diagrams may represent different views on a system, there is no mechanism to define the interconnections or dependencies among the diagrams describing a system. In other words, there are too many places to capture information (in the large number of available diagrams), and too few ways to show relationships between the diagrams (Berkenkotter 2003). Moreover, recent history tells us that the benefits of UML are unlikely to be appreciated by upper-level management and discipline-specific engineers -- instead, issues need to be explained in terms with which they are already familiar (Fogarty and Austin 2009).

These gaps will not be bridged unless a method is found to use UML (and its extensions) in concert with discipline-specific models and notations (e.g., visualization of requirements; block diagrams; two- and three-dimensional engineering schematics). Therefore, the key tenet of the proposed work is that **end-to-end development of engineering systems will occur through multiple models of visualization networked together.** Looking ahead, there will still be a need for development of web-centric, graphically driven, computational platforms dedicated to system-level planning, analysis, design and verification of complex multidisciplinary engineering systems. These environments will employ semantic descriptions of application domains, and use ontologies to enable validation of problem domains and communication (or mappings) among multiple disciplines. The abstraction of multiple disciplines to properly annotated information representations and reuse of previous work at all levels of development will be essential. Present-day systems engineering methodologies and tools are not designed to handle projects in this way.

State-of-the-Art Requirements Modeling

At the system level, designs are viewed as collections (e.g., networks and hiérarchies) of large, arbitrarily complex functional units that form the major components of a system. Designers need to identify components/objects, their attributes and methods, and interfaces and relationships to external entities. To maximize the quality of connunication in team-based development of projects, participants should be able to view design data/information in a manner with which they are familiar, and easily understand connectivity relations and transitions among viewpoints, and the rationale for establishing the connections in the first place. Unfortunately, state-of-the-art capability in requirements modeling and visualization fall short of this vision.

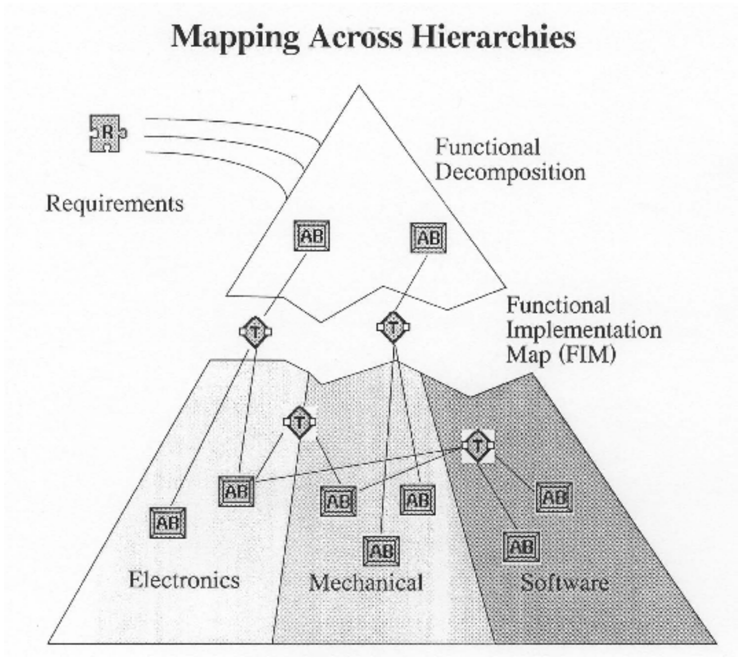


Figure 1. Modeling Transitional Mappings Across Hierarchies in SLATE

As a case in point, the IBM Teamcenter (SLATE) Requirements Tool is based upon very good data representations for traceability links (complying and defining links), and connecting cause-and-effect relationships among abstraction blocks (ABs) in multiple viewpoints (i.e., TRAMS == translational mappings). TRAMs work in terms of connecting source ABs to destination ABs, and source-to-destination and destination-to-source pathways. Figure 1 shows, for example, the use of TRAMs to link requirements, electrical, mechanical and software viewpoints. The underlying graphical support is weak in the sense that all design entities are simply referred to as abstraction blocks (ABs). Moreover, to date, no one has been able to figure out how to actually organize and visualize the subsystem viewpoints on a computer as illustrated in Figure 1. This leaves a non-systems engineer in the dark, providing little visual assistance in understanding how requirements influence design objects that they actually understand, and in understanding how elements in one domain of engineering are affected by concerns in a different engineering domain. To overcome these limitations we need a better representation of individual objects (requirements, abstraction blocks, and so forth) and the linkage of those entities to the overall architectural design.

Proposed Approach to Traceability

The upper half of Figure 2 shows a simplified representation for how requirements are connected to design elements in state-of-the-art traceability (i.e., traceability links connect requirements directly to design objects). State-of-the-art traceability mechanisms portray that ``this requirement is satisfied by that design object (or group of design objects)''. Or alternatively, looking backwards, ``this design object is here because it will satisfy that design requirement.''

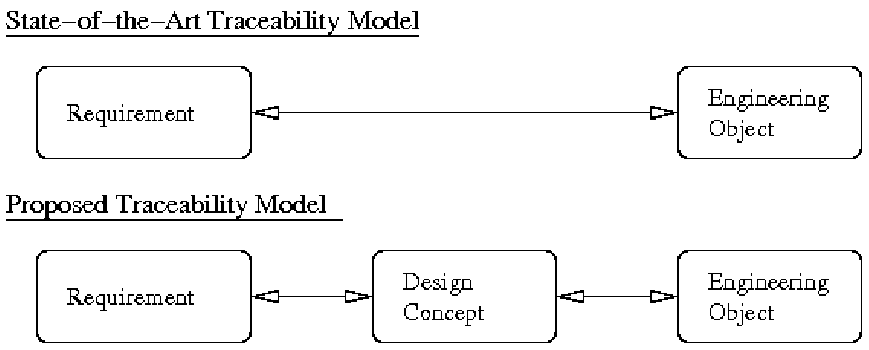


Figure 2. Simplified View of State-of-the-Art Traceability and the Proposed Model

The lower half of Figure 2 shows the proposed model that will be explored in this work. Instead of directly connecting requirements directly to design objects, a new node called ``Design Concept" will be embedded in the traceability link. Assembly of traceability links will be conducted by asking ``what concept should be applied to satisfy this requirement?" Solutions to this question establish links between requirements and design concepts. We assume that the design itself will correspond to the application of appropriate concepts. Thus, the links between design concepts and engineering objects represents an actual implementation of concepts.

From a validation and verification viewpoint, the key advantage of the proposed model is that software for ``design rule checking" can be embedded inside the design concepts module. Thus, rather than waiting until the design has been fully specified, this model has the potential for detecting rule violations at the earliest possible moment. Moreover, if mechanisms can be created to dynamically load design concept modules into computer-based design environments, then rule checking can proceed even if the designer is not an expert in a particular domain.

From a modeling and visualization standpoint, this approach opens the door to improved methods for the visualization of requirements with respect to design objects. In an ideal setting, the latter should be visualized using a notation familiar to the engineer (e.g., a mechanical engineering drawing).

Ontologies and Ontology-Enabled Computing

An ontology is a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic ((Gomez-Perez 2004, Hendler 2001, Staab 2000). Ontologies are needed to facilitate communication among people, among machines, and between humans and machines. Instead of creating a system through the integration of data, the proposed approach follows an approach of creating systems through the application and integration of concepts.

System and sub-system evaluation will depend on both the concept and the data used in its implementation (e.g., an area constraint will depend on geometry). To ensure that system-level designs are faithful representations of both the stakeholder needs and the capabilities of the participating application domain(s), ontology models need to be accurate, complete, conflict free and minimal (i.e., no redundancy) (Shanks 2003). Accuracy means that models need to accurately represent the semantics of the participating application domains, as perceived by the project stakeholders. To reduce the likelihood of conflicts during model updates, models should not contain redundant semantics. The ontology community makes a distinction between ontologies that are taxonomies and those that model domains in depth, applying restrictions on domain semantics (Gomez-Perez 2004). So-called lightweight ontologies include concepts, concept taxonomies, relationships between concepts, and properties of the concepts. Heavyweight ontologies add axioms to lightweight ontologies -- axioms serve the purpose of adding clarity to the meaning of terms in the ontology. They can be modeled with first-order logic. Top-level ontologies describe general concepts (e.g., space, connectivity, etc.). Domain ontologies describe a vocabulary related to a particular domain (e.g., building architecture, plumbing, etc.). Task ontologies describe a task or activity. Application ontologies describe concepts that depend on both a specific domain and task. These ontologies might represent user needs with respect to a specific application. Because a unified theory for system validation does not exist at this time, present-day procedures for design rule checking tend to focus on small snippets of the system model functionality, and are achieved in several ways: (1) consistency checking, (2) connectivity analysis, and (3) model analysis on a global basis, based upon graph-theoretic techniques.

Ontology-Enabled Computing. To provide for a formal conceptualization within a particular domain, and for computers to share, exchange, and translate information within a domain of discourse, an ontology needs to accomplish three things (Liang 2004): (1) Provide a semantic representation of each entity and its relationships to other entities; (2) Provide constraints and rules that permit reasoning within the ontology; and (3) Describe behavior associated with stated or inferred facts. Items 1 and 2 cover the concepts and relations that are essential to describing a problem domain. Items 2 and 3 cover the axioms that are often associated with an ontology. Usually, axioms will be encoded in some form of first-order logic.

Semantic Web. This project assumes that advances in ontology-enabled design and development will occur in parallel with advances in the Semantic Web.

In his original vision for the World Wide Web, Tim Berners-Lee described two key objectives: (1) To make the Web a collaborative medium; and (2) To make the Web understandable and, thus, processable by machines. During the past decade the first part of this vision has come to pass -- today's Web provides a medium for presentation of data/content to humans. Machines are used primarily to retrieve and render information. Humans are expected to interpret and understand the meaning of the content.

The Semantic Web (Berners-Lee 2001, Hendler 2001) aims to give information a well-defined meaning, thereby creating a pathway for machine-to-machine communication and automated services based on descriptions of semantics (Geroimenko 2003). Realization of this goal will require mechanisms (i.e., markup languages) that will enable the introduction, coordination, and sharing of the formal semantics of data, as well as an ability to reason and draw conclusions (i.e., inference) from semantic data obtained by following hyperlinks to definitions of problem domains (i.e., so-called ontologies).

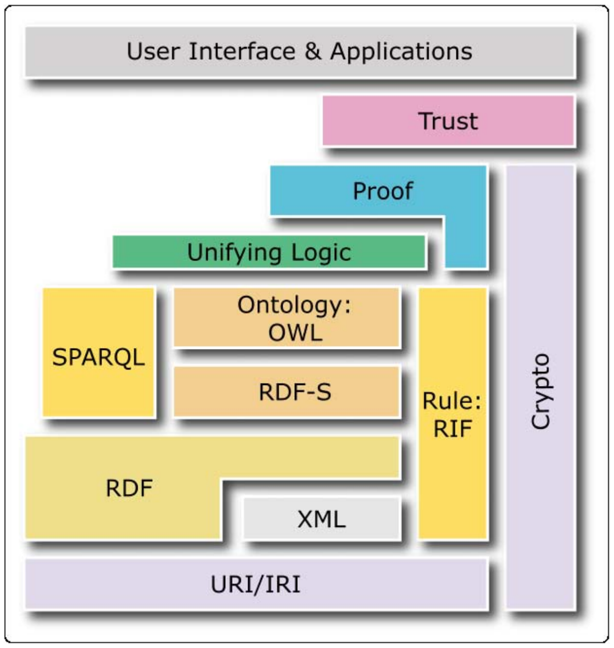


Figure 3. Technologies in the Sematic Web Layer Cake

Figure 3 describes the infrastructure that will support this vision (Berners-Lee 2000). The Resource Description Framework (RDF) defines a standard for describing the relationships between objects and classes in a general but simple way. Class relationships and statements about a problem domain are expressed in DAML+OIL (DARPA Agent Markup Language) and more recently, the Web Ontology Language (OWL) (Webont 2003).

Representation of Ontologies with UML Class Diagrams. From a systems engineering perspective, the key advantage in modeling design concepts with Semantic Web languages such as RDF, DAML and OWL is that software tools have been developed for logical reasoning with relationships and rules implied by ontologies, and for evaluation of assertions. See Figure 5. Unfortunately, at this time RDF, DAML and OWL lack a standard representation for visualizing concepts expressed in these languages.

A practical way of overcoming this shortcoming is to use UML class diagrams -- actually, graph structures of UML schema -- in lieu of a formal ontology. UML is well defined and has a community of millions of users. UML class diagrams can be used for representing concepts (and their attributes), and relations between concepts (e.g., knowledge reflecting performance, legal and economic restrictions). Basic relationships, such as inheritance and association can be modeled. Axioms (i.e., additional constraints) can be represented in the Object Constraint Language (OCL).

This idea is not new. The close similarity of DAML and UML has been established by Cranefield and co-workers (Cranefield 2001a, 2001b). For example, both DAML and UML have a notion of a class which can have instances. The DAML notion of a subClassOf is essentially the same as the

UML notion of specialization/generalization. Thus, UML qualifies as a visual representation for ontologies (Baclawski 2001). Moreover, tools are starting to emerge for the automated transformation of ontologies to UML. See, for example, descriptions of the tool DUET in Kogut et al. (Kogut 2002).

Meta Model for the Proposed Approach

Meta Models and Meta-Meta Models. Most engineers think of UML as simply a diagramming notation for the high-level, albeit informal, specification of system structure and behavior. UML is, in fact, based on well-defined language concepts specified in terms of meta-models and meta-meta-models. Diagrams are one representation of the UML language concepts. An equivalent XML representation also exists.

A meta-model describes information about models. Meta-meta-models describe information about meta-models.

Figure 4 shows the pathway from meta-meta-models to meta-models to models and implementation of engineering systems. Key points:

1. The meta-meta-model (also known as the UML meta-model) is a model that describes the UML language -- specifically, it describes classes, attributes, associations, packages, collaborations, use cases, actors, messages, states, and all the other concepts in the UML language.

2. UML-like diagrams express concepts and relationships among concepts suitable for creating a design. These diagrams serve as a meta-model for the development of potentially acceptable designs.

3. The UML diagrams themselves are created from diagram éléments having well-defined semantic meaning. The set of diagram elements (e.g., notations for inheritance, aggregation, and so forth) form a meta-meta model.

4. Requirements are satisfied by applying a concept expressed in the meta-model. The activation of a concept results in an object in the design model. The latter is shown on the bottom right-hand side of Figure 5.

A meta-model is a precise definition of the constructs and rules needed for creating semantic models. Models are the first level of abstraction from "systems of interest" to the modeler. Meta-models are the second level of abstraction – the items of interest at this level are the elements, rules and meaning of the modeling constructs themselves. Meta-meta-models define a language in which meta-models may be expressed.

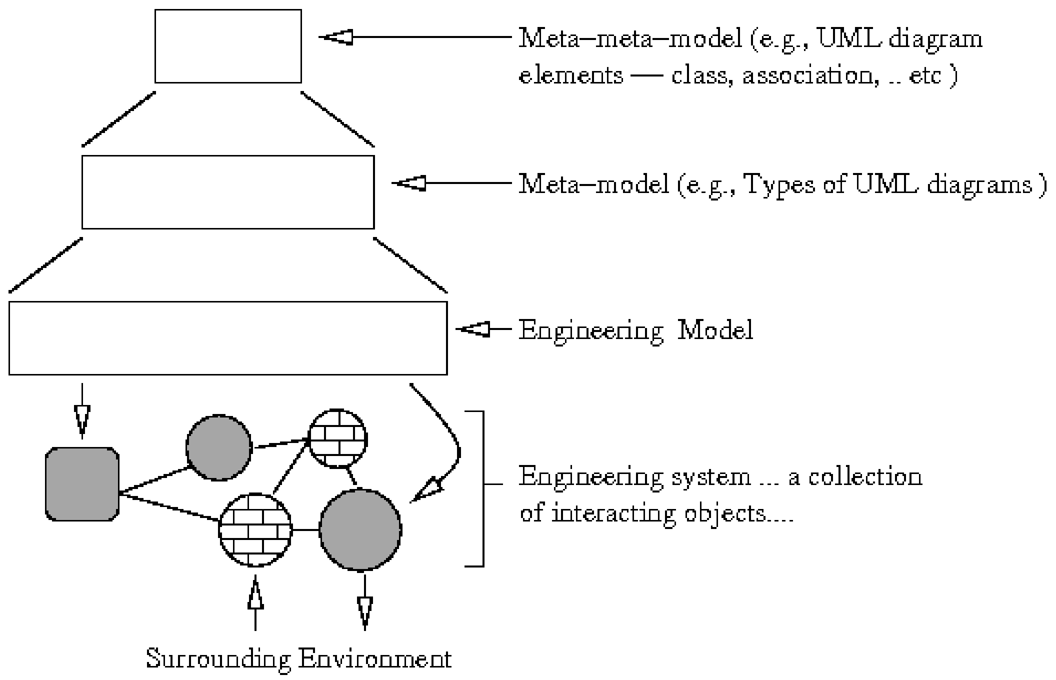

Meta–meta–model (e.g., UML diagram elements — class, association, .. etc)
Meta–model (e.g., Types of UML diagrams)
Engineering Model
Engineering system ... a collection of interacting objects....
Surrounding Environment

Figure 4. Pathway from Meta-meta-models to Engineering Models and Systems (Source, Wie, 1998)

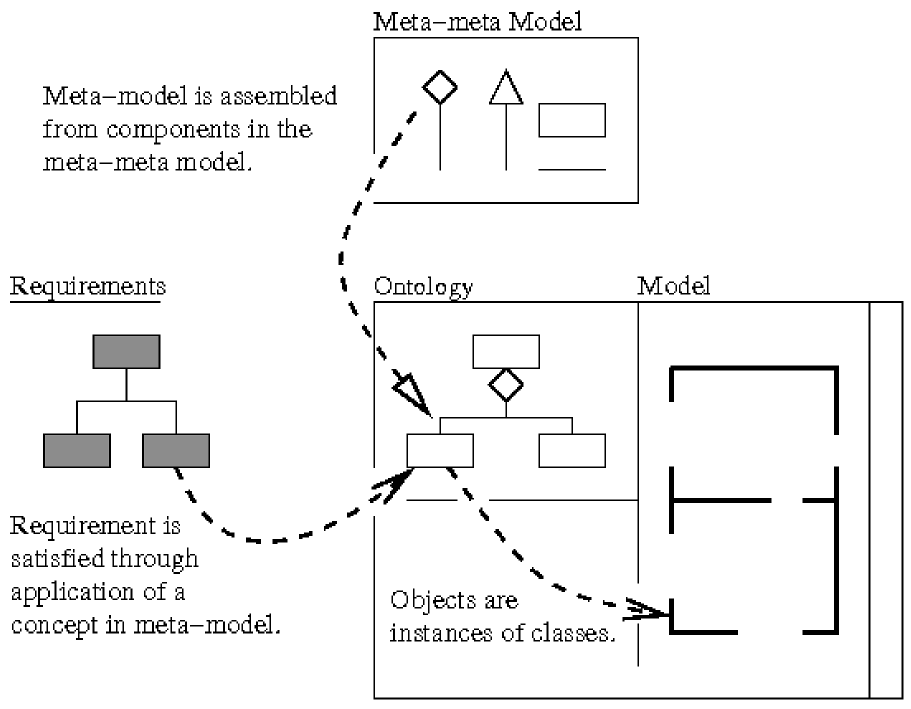

Meta–meta Model
Meta–model is assembled from components in the meta–meta model.
Requirements
Ontology
Model
Requirement is satisfied through application of a concept in meta–model.
Objects are instances of classes.

Figure 5. Meta-Model for the Proposed Approach

Software Architecture Design

Software architecture design is concerned with the sélection and configuration of major software components and their connectivity. For this context, connectivity means : (1) linking of requirements to UML classes (i.e., the ontology), and (2) linking of UML classes to objects in the engineering model. As illustrated in Figure 6, we expect that software implémentations will operate as a network of loosely coupled systems, connected only by traceability mechanisms and interfaces for communication of évents and required data for évaluation of design rules.

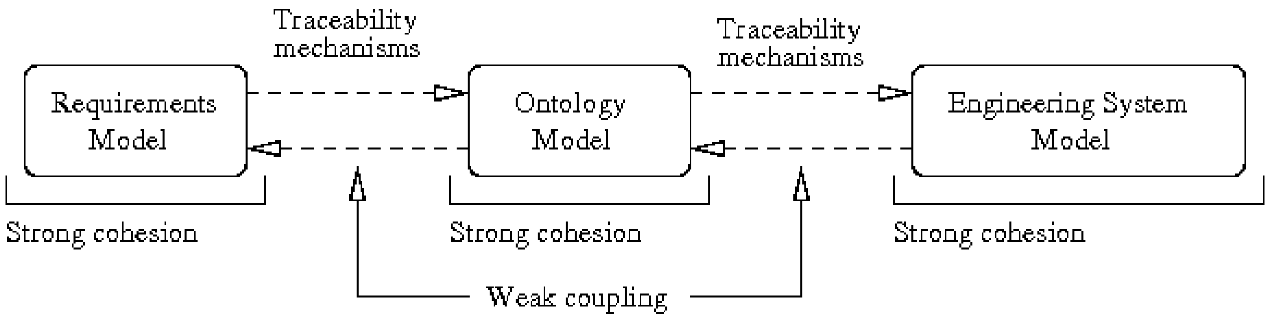


Figure 6. Overview of System Architecture

The software architecture for the prototype implementation makes exclusive use of two technologies: (1) the JavaBeans framework for establishing graphs of listener-driven events using the DEM; and (2) the Violet UML Editor graphical user interface framework.

Graphical User Interface Design. Figure 7 shows the layout of windows in the prototype software implémentation and mechanisms for storage of requirements, ontologies and engineering models in an XML data format. The graphical user interface is a composition of three panels, a requirements panel containing the table of requirements, a UML diagram panel for the application ontology, and an engineering model panel containing the model of the system. The panel assembly implements the notion of a reactive design environment, where users can query the system to establish relationships among the requirements, ontologies and physical design entitites.

Delegation Event Model. Traceability connectivity and communication mechanims are handled by the Java Delegation Event Model (DEM). The DEM is based on the Publish-Subscribe design pattern. The main objectives of Publish-Subscribe are to provide a method of signaling from a publisher to subscribers and to provide a method to dynamically register and deregister subscribers with a publisher. Publishers generate and send events, and subscribers register or subscribe to those events from the publishers. When a publisher sends out or publishes an event, all subscribers interested in that event are notified. The DEM refers to publishers as *event sources* and subscribers as *event listeners* (Larman 1999).

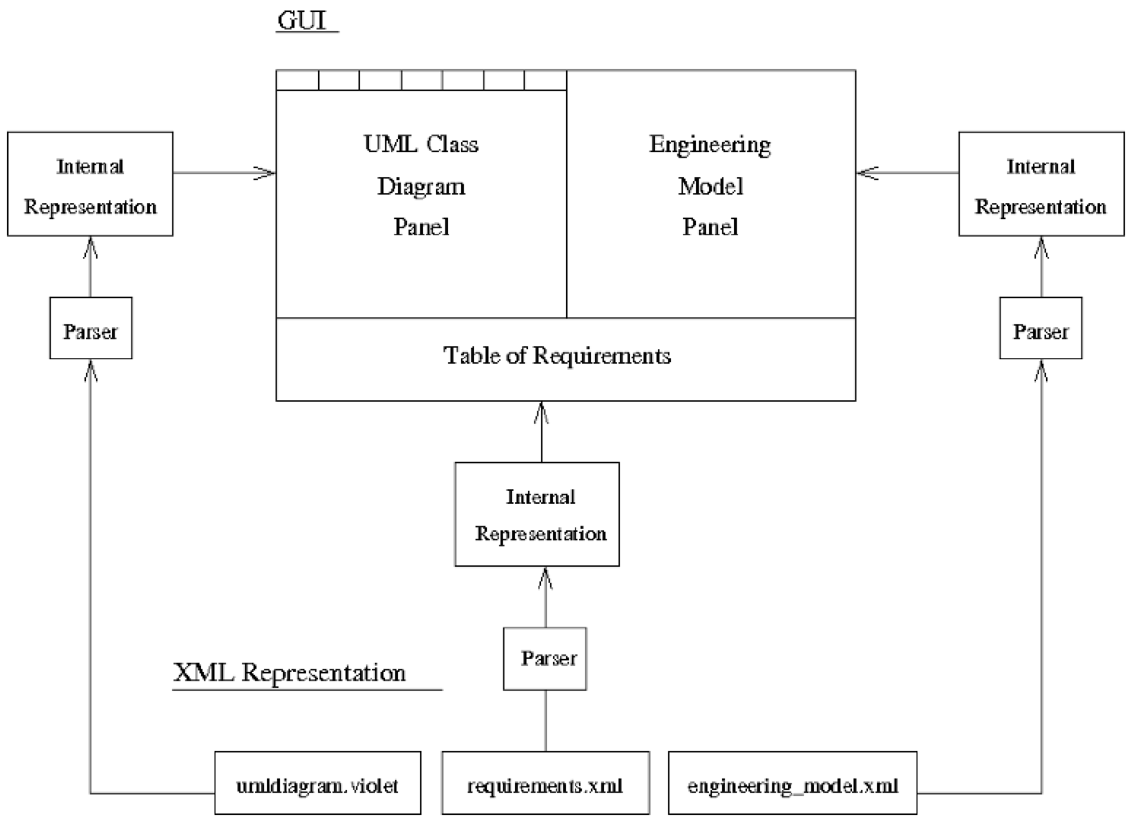


Figure 7. Graphical User Interface Layout and Connection to XML Persistent Storage

Ontology-to-Engineering Model Connectivity. Standard implementations of computational support for UML diagramming have the goal of providing end-users with the ability to easily create static diagrams. Here, in contrast, UML classes and class diagrams serve the dual role of: (1) representing domain ontologies and (2) enabling linkages between requirements and engineering objects. Computational support has the goal of providing executable services for design traceability and design rule checking.

Figures 8 and 9 show the step-by-step procedure for development, implementation and operation of ontology-enabled traceability in a design specification setting. The implementation needs to support: (1) Definition of relationships (e.g., one-to-one, one-to-many, etc.). (2) Management of relationships (e.g., create, trace, and remove) and (3) Inquiry for availability of services. Looking forward (see Figure 8), each specification class will store tables of references to objects in the physical design. Looking backward (not shown), these references will be connected to one or more design requirements. Figure 9 shows the pathway of development for the processing of user events and design rule checking. The main point to note is that the ontology is not just a pictorial representation; rather it becomes an ontology processing machine that accepts registration of requirements and design object interest in events, and supports design rule checking. A full-scale implementation would also show dependencies among ontologies – the exact details on how this should work (perhaps with three-dimensional graphics) are currently being worked out.

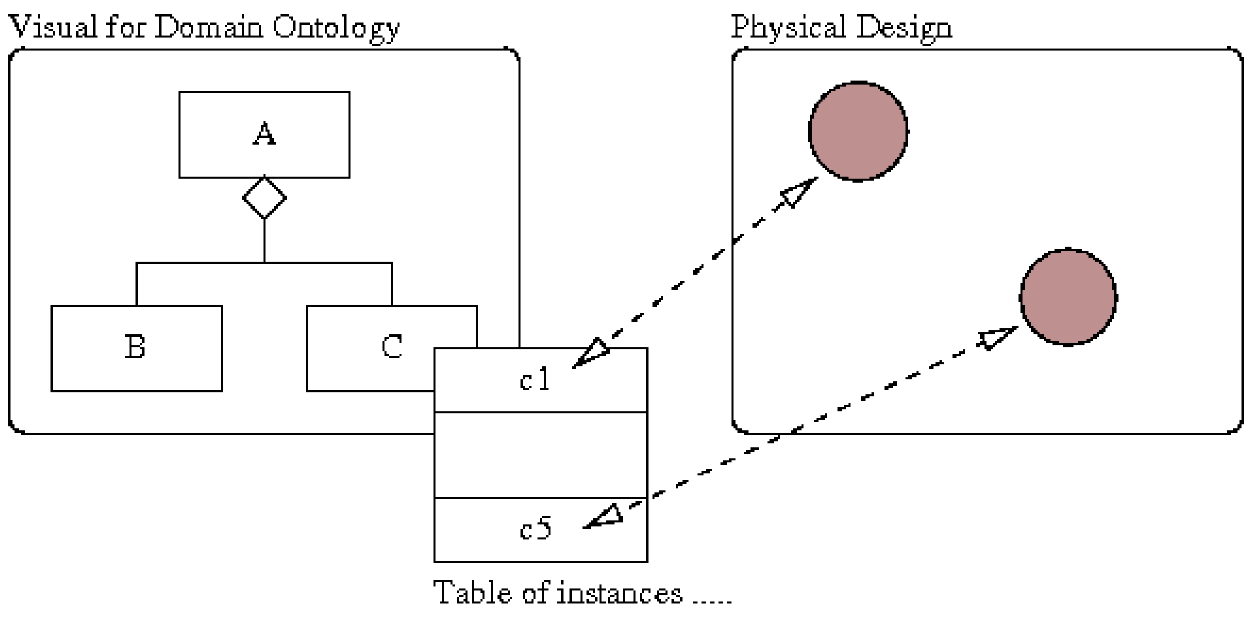


Figure 8. Connectivity Between the Ontology and Physical Models

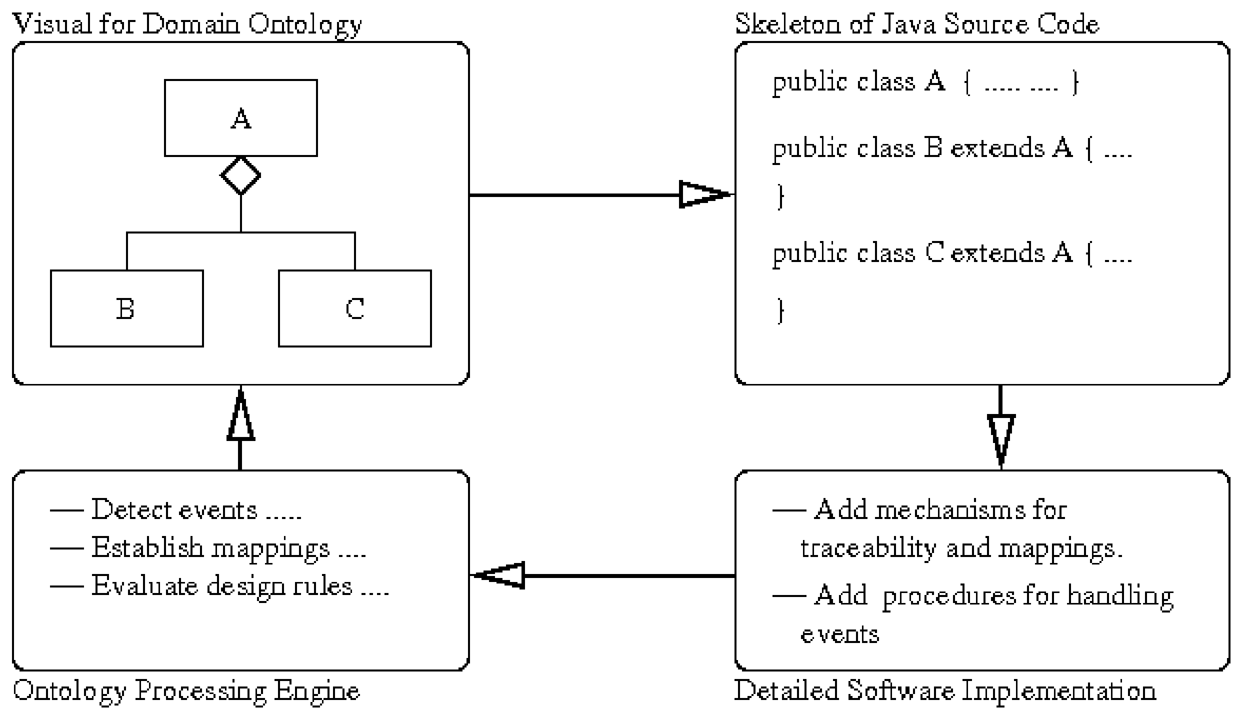
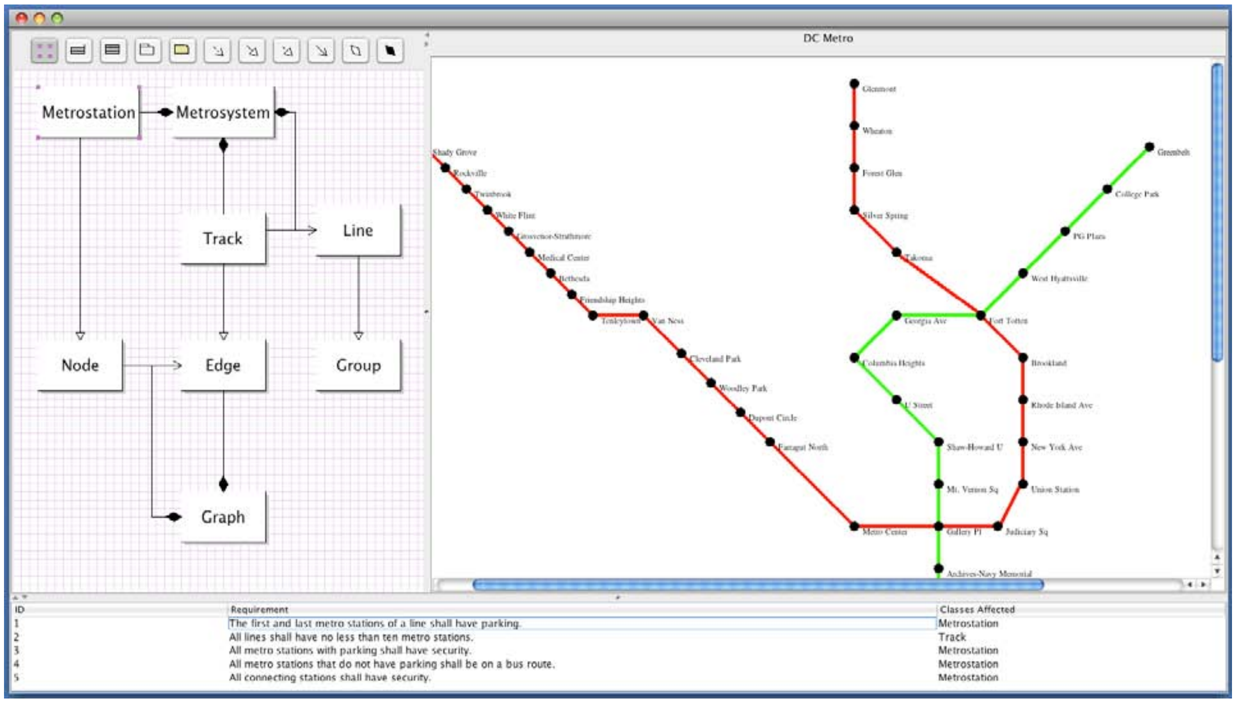

Visual for Domain Ontology
A
B
C
Skeleton of Java Source Code
public class A { .... }
public class B extends A { ....
}
public class C extends A { ....
}
— Add mechanisms for traceability and mappings.
— Add procedures for handling events
Detailed Software Implementation
— Detect events .....
— Establish mappings ....
— Evaluate design rules ....
Ontology Processing Engine

Figure 9. Step-by-step Implementation of Ontology Processing Machine

Application : Washington DC Metro System

This section presents our first prototype application of ontology-enabled traceability. The Washington DC Metro System is the second largest rail transit system in the United States. It serves a population of 3.5 million people with more than 200 million passenger rides per year. As of 2006, there were 86 metro stations in service and 106.3 miles of track.

Requirements-Ontology-Engineering Software Prototype. Figure 10 is a screendump of the Washington DC Metro System Requirements-Ontology-Engineering Model interface.



ID	Requirement	Classes Affected
1	The first and last metro stations of a line shall have parking.	Metrostation
2	All lines shall have no less than ten metro stations.	Track
3	All metro stations with parking shall have security.	Metrostation
4	All metro stations that do not have parking shall be on a bus route.	Metrostation
5	All connecting stations shall have security.	Metrostation

Figure 10. Screendump of the Washington DC Metro System

The software prototype has a user interface and XML input/output consistent with the spécifications of Figure 7. Component connectivity relationships are modeled with graph data structures. Metro station and group objects are identified by their name. A symbol table is employed for fast storage and retrieval of named objects. XML import/export is handled by JAXP, the java interface for XML processing with DOM parsers.

The métro system design is modeled with only five requirements : (1) The first and last métro stations of a line shall have parking, (2) All lines shall have no less than ten métro stations, (3) All métro stations with parking shall have security, (4) All métro stations that do not have parking shall be on a bus route, and (5) All connecting stations shall have security. Requirements 1 and 3 through 5 are satisfied by apply concepts in the MetroStation class. Requirement 2 is satisfied by apply concepts in the Track class/ontology.

The top left-hand panel shows the métro system ontology represented in a UML class diagram format. The ontology diagram serves two perspectives. From a mathematical standpoint, the transportation network is simply as a graph of nodes connected by edges. A node can be characterized by its name and geographical position. Well known algorithms exist for questions of reachability and routing. A transportation viewpoint builds upon the mathematical viewpoint by adding attributes and conveniences suitable for transportation engineering. Métro stations are modeled as graph nodes plus information on parking and security. Notions of a transportation track correspond to edges in the graph. To simplify and facilitate navigation, groups of tracks are organized into color-coded line abstractions (e.g., riders talk about catching a green line train to the College Park Metro Station, but in reality neither the trains nor track are actually painted green).

Listener-Driven Event Model for Requirements Traceability. The requirements, ontology, and engineering entities are connected and communicate through the use of a listener-driven event model. Individual requirements register with the UML classes containing the concepts relevant to their eventual satisfaction. Then, in turn, individual UML class nodes register with individual and groups of design objects that a ultimately responsible for implementing a requirement. Pathways of traceability also begin with objects in the engineering model and work their way back to individual (or groups) of requirements. The result is a mixture of one-to-many and many-to-many relationships in a graph of bi-direxctional traceability relations.

User Interaction with the Requirements Panel. When single-clicking on a requirement, the classes that are affected by that requirement are notified of the event. The classes in the UML diagram are highlighted and the items in the engineering drawing are highlighted, because they are registered to listen to the single-click évent from the requirement. Double clicking a requirement triggers the verificatio of that requirement against the engineering model. For example, the first requirement (end of line métro stations shall have parking) can be checked by simply double clicking on the requirement. Two things happen. First, a smal popup window will indicate whether or not the requirement has been violated. And second, all of the associated ontology components and physical design objects that are part of the rule checking procèss will be highlighted.

User Interaction with the UML and Engineering Model Panels. When mousing-over a UML class node, the engineering drawing objects and requirements that are registered to listen to that event are notified. The objects in the engineering drawing are highlighted and all requirements that affect the class are highlighted because they are registered to listen to the mouse-over event from the class node. For example, when the cursor is positioned over the Metrostation class node, all of the Metro station nodes in the engineering drawing are highlighted, as are all of the requirements that depend on class Metro Station for their satisfaction. Similar behavior occurs when the cursor is positioned over an object in the engineering model/drawing.

Conclusions and Future Work

Conclusions. This project is motivated by our belief that the likelihood of serious system failures can be mitigated with traceability modeling that supports validation and/or vérification procedures early in the development lifecycle. Traceability models need to link together multiple models of visualization. The key contribution of this work is preliminary evaluation of a new type of traceability link, where design concepts are inserted between the already connected requirements and engineering objects. Traceability relationships between requirements, design concepts and engineering objects may be arbitrarily complex, possibly forming a very large graph structure. Procedures for establishing these links and responding to external user events need to be efficient and scalable. Here we have shown that UML class diagrams and listener-event models provide a suitable framework for creating a variety of traceability relationships (e.g., one-to-one, one-to-many, etc) A key benefit in this new type of traceability link is that rule checking procedures may be embedded into design concept nodes. Since individual design concept nodes are part of a larger ontology, rule checking procedures should apply across all projects where the ontology is applicable. Of course, the details of rule evaluation may differ from one technology to the next.

Future Work. Our ontology-enabled traceability model is now being extended in two directions. We are adding timetable-driven train behavior to the Washington DC Metro system model. This extension opens the possibility of traceability connections between functional/performance requirements and individual states of finité-state machine behavior. The small table of requirements will be replaced by PaladinRM, an interactive java-based tool for working with large graphs of engineering requirements (Austin et al. 2006a). In the second direction of work, we are investigating the feasibility of replacing UML diagrams with the Web Ontology Language (OWL) and reasoning procédures driven by the Semantic Web Rule Language (SWRL).

References

Austin M., Mayank V., and Shmunis N. 2006a. PaladinRM: Graph-based Visualization of Connectivity Relationships Organized for Team-Based Design, *Systems Engineering*, Vol. 9, No. 2, pp. 129-145, May.

———. 2006b. Ontology-Based Validation of Connectivity Relationships in a Home Theatre System, *International Journal of Intelligent Systems*, Vol 21., No. 10, pp. 1111-1125, October.

Baclawski K., Kokar M.K., Kogut P., Hart L., Smith J., Holmes W., Letkowski J., and Aronson M.L. 2001. Extending UML to Support Ontology Engineering for the Semantic Web, *UML 2001*.

Berners-Lee T., Hendler J., and Lassa O., 2001. The Semantic Web, *Scientific American*, pp. 35-43.

Cranfield S. 2001. Networked Knowledge and Representation and Exchange using UML and RDF, *Journal of Digital Information*, Vol. 1, No. 8.

———. 2001. UML and the Semantic Web, In the Proceedings of the International Semantic Web Working Symposium, Palo Alto, CA.

Faison T., 2006. Event-Based Programming: Taking Events to the Limit, Apress, New York.

Fogarty K., Austin M. 2009. Systems Engineering and Traceability Applications of the Higraph Formalism, *Systems Engineering*, Vol. 12, No. 2, pp. 117-140.

Gomez-Perez. A., Fernandez-Lopez. M., and Corcho O. 2004. Ontological Engineering, Springer.

Hendler J. 2001., Agents and the Semantic Web, *IEEE Intelligent Systems*, March/April, pp.

30-37.

Horstmann C. 2006., Object-Oriented Design and Patterns: Second Edition, John-Wiley and Sons, New York. Note: see, in particular, pages 334-352.

Jackson, D. 2006. Dependable Software by Design, Scientific American, Vol. 294, No. 6.

Jones N., 2004. Flawed Drawings caused Spacecraft Crash: Upside-Down Switches Stopped Parachutes from Opening, *Source:* *news@nature.com*.

Kogut P., Cranefield S., Hart I., et al. 2001, UML for Ontology Development, The Knowledge Engineering Review, Vol. 17, No. 1.

Liang V.C., Paredis C.J.J. 2004. A Port Ontology for Conceptual Design of Systems, Transactions of the ASME, Vol. 4, September.

Sawyer K., 1999. Engineers' lapse leads to loss of Mars Spacecraft: Lockheed didn't tally Metric Units, *Washington Post*.

Shanks G., Tansley E., Weber R. 2003. Using Ontology to Validate Conceptual Models, Communications of the ACM, Vol. 46, No. 10., pp. 85-89.

Staab S., Maedche A. 2000. Ontology Engineering beyond the Modeling of Concepts and Relations, In Benjamins R.V., Gomez-Perez A., Uschold M., editors. Proceedings of the 14th European Conference on Artificial Intelligence, Workshop on Applications of Ontologies and Problem Solving Methods.

Unified Modeling Language (UML), 2003. See http://www.omg.org/uml.

Web Ontology Language (OWL), 2003. See http://www.w3.org/TR/owl-ref/

Whitney D.E. 1996. Why Mechanical Design cannot be like VLSI Design, Research in Engineering Design, Vol. 8, pp. 125-138.

Biographies

Mark Austin is an Associate Professor of Civil and Environmental Engineering at the University of Maryland, College Park, with a joint appointment in the Institute for Systems Research (ISR). Mark is Director of the Master of Science in Systems Enginering (MSSE) Program at ISR. Mark has a Bachelor of Civil Engineering (First Class Honors) from the University of Canterbury, Christchurch, New Zealand, and M.S. and Ph.D. degrees in Structural Engineering from the University of California, Berkeley.

Cari Wojcik is a Systems Engineer in the Civil Security and Response Program, Raytheon Integrated Defense Systems, Portsmouth, Rhode Island. She has a MBA from the University of Rhode Island (2009), a Masters of Systems Engineering from the University of Maryland (2006), and an undergraduate degree in Computer Engineering from Virginia Tech (2004).