THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2010-20

# Methodology and System for Ontology-Enabled Traceability: Pilot Application to Design and Management of the Washington D.C. Metro System

Mark Austin
Cari Wojcik

The Institute for Systems Research

A. JAMES CLARK SCHOOL OF ENGINEERING

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the A. James Clark School of Engineering. It is a graduated National Science Foundation Engineering Research Center.

www.isr.umd.edu


**ISR Technical Report 2010-21**

# Methodology and System for Ontology-Enabled Traceability: Pilot Application to Design and Management of the Washington D.C. Metro System

**By Mark Austin[1] and Cari E. Wojcik[2]**

Last modified: December 20, 2010

[1]Associate Professor, Department of Civil and Environmental Engineering, and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA. E-mail: austin@isr.umd.edu

[2]Civil Security and Response Programs, Raytheon Integrated Defense Systems, Portsmouth, RI 02871, USA. (Formerly, MSSE Student at the University of Maryland). E-mail: cari_e_wojcik@raytheon.com

# Abstract

This report describes a new methodology and system for satisfying requirements, and an architectural framework for linking discipline-specific dependencies through interaction relationships at the meta-model (or ontology) level. In state-of-the-art traceability mechanisms, requirements are connected directly to design objects. Here, in contrast, we ask the question: What design concept (or family of design concepts) should be applied to satisfy this requirement? Solutions to this question establish links between requirements and design concepts. Then, it is the implementation of these concepts that leads to the design itself. These ideas are prototyped through a Washington DC Metro System requirements-to-design model mockup. The proposed methodology offers several benefits not possible with state-of-the-art procedures. First, procedures for design rule checking may be embedded into design concept nodes, thereby creating a pathway for system validation and verification processes that can be executed early in the systems lifecycle where errors are cheapest and easiest to fix. Second, the proposed model provides a much better big-picture view of relevant design concepts and how they fit together, than is possible with linking of domains at the model level. And finally, the proposed procedures are automatically reusable across families of projects where the ontologies are applicable.

**Keywords:** systems engineering, traceability, ontologies.

# Contents

# Chapter 1

# Introduction

## 1.1 Problem Statement

Good engineering design solutions are nearly always required to balance the need for system functionality and maximum performance against limitations on cost. Common requirements include the need for reliable system operation in a wide range of environments, and ease of accommodation for future technical improvements and changes to requirements. And due to their large overall size, good engineering design solutions are nearly always developed by teams of engineers. Figure 1.1 shows, for example, a hypothetical situation where high-level project requirements are organized for team development, and project requirements are imported from external sources, in this case, the Environmental Protection Agency (EPA). Methodologies for the team development of system-level architectures need to support: (1) Partitioning the design problem into several levels of abstraction and viewpoints suitable for concurrent development by design teams; (2) Coordinated communication among design teams; (3) Integration of the design team efforts into a working system; and (4) Evaluation mechanisms that provide a designer with a critical feedback on the feasibility of system architecture, and make suggestions for design concept enhancement. It is the responsibility of the systems engineer to gather and integrate subsystems and to ensure that every project engineer is working from a consistent set of project assumptions. This requires an awareness of the set of interfaces and facilities to which the system will be exposed. Systems engineers are also responsible for trade studies to find a good balance in competing (design and business) criteria. Again, this capability requires an awareness of the connectivity mechanisms among all systems entities.

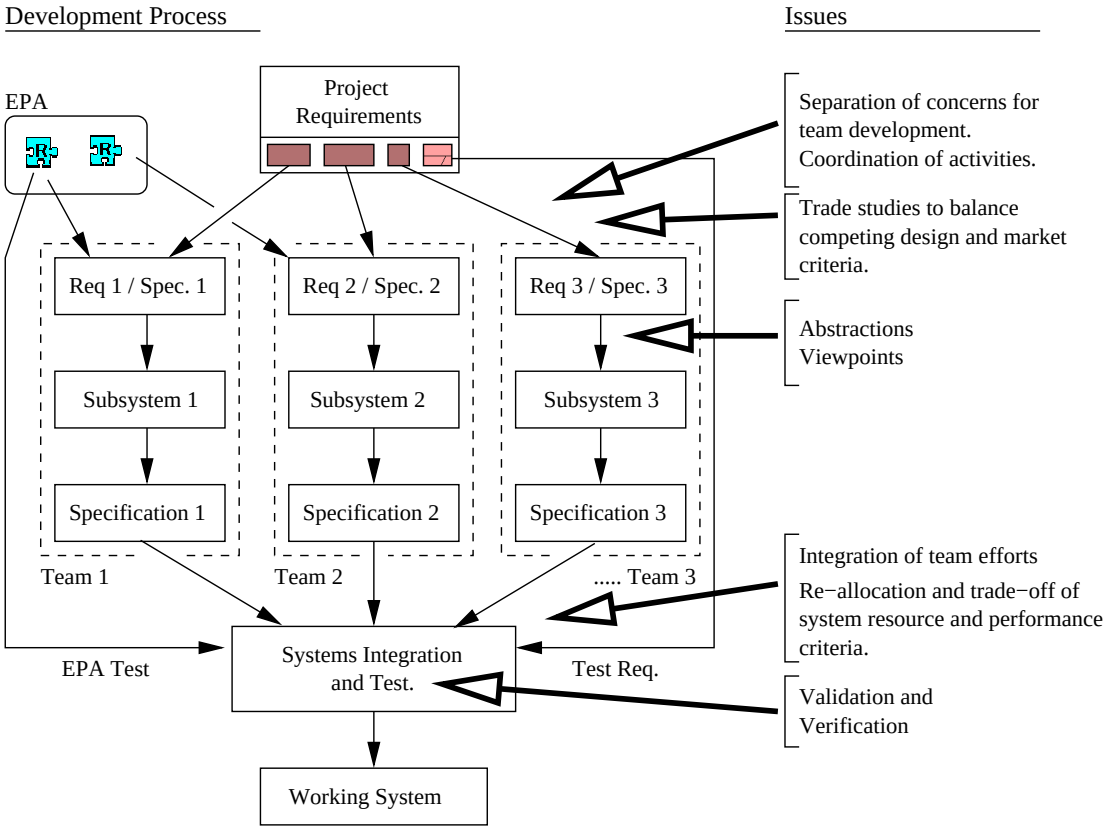



Figure 1.1: Development process and key issues in the team-based development of engineering systems.

**Transition from Industrial- to Information-Age Capability.** Nowadays, the systematic consideration of these design concerns is complicated by a general trend toward the replacement of industrial-age systems with information-age systems, the latter being a lot more instrumented and interconnected than their predecessors [29].

Industrial-age systems are mechanical and electro-mechanical systems that date as far back as the late 1800s [60]. As illustrated in Figure 1.2, established approaches to industrial-age development deal with complexity through a systematic separation and decomposition of design concerns into weakly coupled hierarchies of simpler discipline-specific problems. While this strategy simplifies design, an analysis of industrial-age system capability reveals that in many cases, limitations on achievable functionality and performance can be attributed to a general inability of humans to sense the surrounding environment, control system behavior, and look ahead and anticipate important events in a manner consistent with high performance and wide ranges of functionality. While humans are good at collecting/sensing data and synthesizing information from it, they are very slow (especially compared to computers) and also easy tire.

Information-age systems are developed under the premise that expanded system functionality and improved performance can be achieved through the use of distributed system structures, concurrent subsystem behaviors, mixtures of centralized and decentralized control and use of technologies that move the boundary representing limits on what is possible [64]. As such, information-age systems correspond to mixtures of hardware, software, and communications, and are often assembled from smaller sub-systems having autonomous behaviors (i.e., so-called system of systems). Within the automobile industry, for example, large-scale mechanical machines (the car of the 1950s) have evolved into networks of computers and electro-mechanical machines on wheels (the car of 2010). The increased use of communications is simply a consequence of the world being more interconnected than 50 years ago. The increased use of sensing means that systems must be capable of collecting and processing large quantities of data and synthesizing relevant information needed for decision making. And the increased use of software allows for programmable system functionality.

**Understanding System Failures.** Generally speaking, information-age systems are required to provide new types of time-critical services, superior levels of performance, and work correctly with no errors.

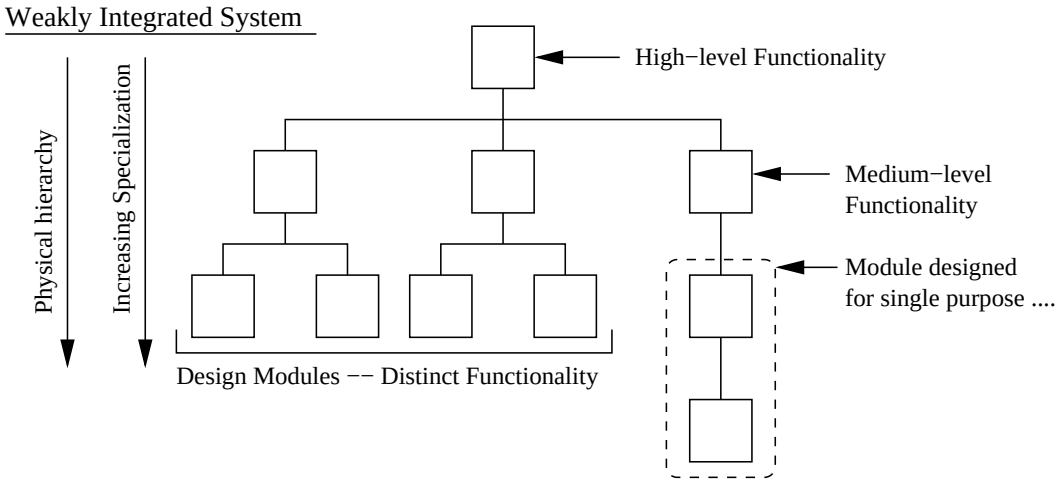



Figure 1.2: Nodal connectivity and functional influence in a weakly-integrated system (adapted from Calvano and John [9]).

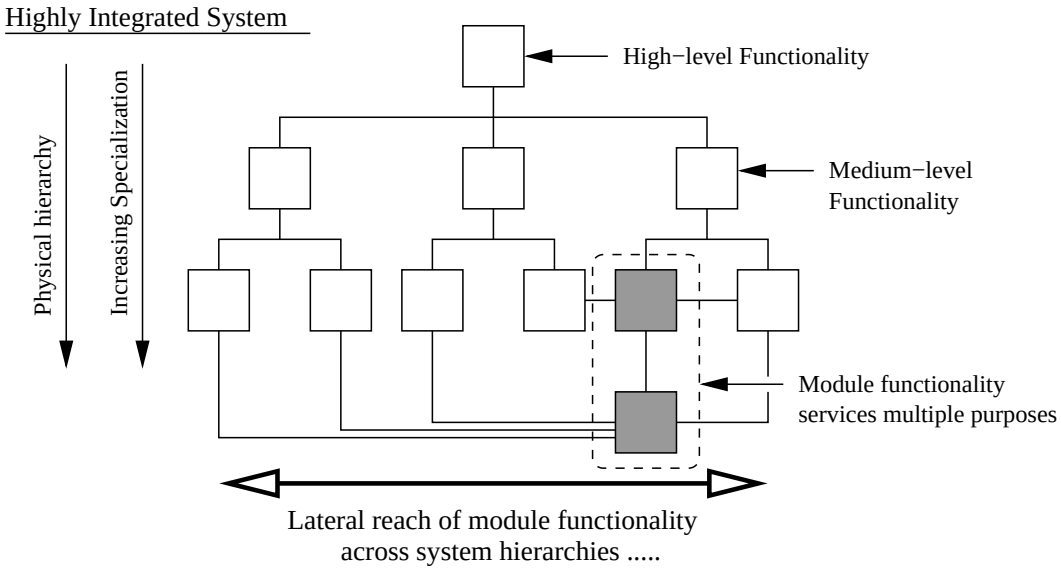



Figure 1.3: Nodal connectivity and functional influence in a highly-integrated system (adapted from Calvano and John [9]).

.

The unfortunate reality is that when new technologies are weaved together to achieve new types of functionality, systems can fail in new and unprecedented ways. The post-event analysis of recent engineering system failures [32, 33, 52] indicates that the underlying cause of catastrophic and expensive failures can vary from major architectural mistakes all the way down to tiny 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). One vexing concern is the increased use of highly-integrated system architectures to extend functionality and improve performance. The underlying motivation for this trend is surprisingly simple: a system will function better when the sub-systems work together as a team rather than independently. Figure 1.3 shows, however, that the key characteristics of integrated system complexity include:

**1.** Lateral influences that dominate hierarchical relationships,

**2.** Cause and effect relationships are not obvious and direct.

A change at almost any level may have system-wide consequences. Influences and impacts of decisions are less predictable and difficult to bound. In some cases the lateral interactions between systems are not well understood. This can lead to surprising patterns of failure across networks. Validating a design for correct functionality and adequate performance is much more difficult than before. As a case in point, correct functionality for software is defined by logic (not differential equations). Not only does the concept of safety factors not apply, but 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 [32] indicate that almost all grave software problems can be traced back to conceptual mistakes made before the programming even started.

**Mechanisms for Keeping the Complexity of Design in Check.** Lessons learned from industry [32, 39, 51] indicate that there are now many automated engineering systems with complexity approaching the point where validation of design correctness will be impossible without mechanisms for pre-deployment reasoning about system requirements and design built into the design process itself. These mechanism include [4, 50]:

1. **Formal Models.** We need ways to capture the design representation and it specification in an unambiguous formal language that has precise semantics.

2. **Abstraction.** Abstraction mechanisms eliminate details that are of no importance when evaluating system (functionality, performance, cost) with respect to a particular viewpoint.

3. **Decomposition.** Decomposition is the process of breaking a design at a given level of hierarchy into subsystems and components that can be designed and verified almost independently.

In established approaches to system design (see Figures 1.4 and 1.5) procedures for "system testing" are executed toward the end of system development. The well-known shortcoming of this approach is the excessive cost of fixing errors. Emerging approaches to system design [39, 55, 61] aim to make validation and integral part of the whole development process, and to maximize the use formal methods and selective use of design abstractions. As illustrated along the right-hand side of Figure 1.5, the goal is to move design processes forward to the point where early detection of errors is possible and system operations are correct-by-construction [55].

### 1.1.1 Response of the Systems Engineering Community

In an effort to improve the accuracy and effectiveness of system-level architectural designs and communication among engineers in the development of engineering systems, the systems engineering community has developed SysML, the Unified Modeling Language (UML) extended and adapted for the needs of Systems Engineers [58, 62]. The underlying motivation for SysML stems from the software engineering community, which has already experienced great success with UML as a representation for informal models of software design. By introducing a variety of new diagram types to SysML that systems engineers need, and removing UML diagram types not of primary importance to systems engineers, the hope is that similar success will occur in systems engineering. The pillars of SysML are visual modeling support for system structure, system behavior, systems requirements and parametric relationships. The UML class diagram (a staple of software engineering) has been removed and, instead, SysML employs a general-purpose block diagram. Mapping relationships (e.g., assignment of functions to structural elements) are handled by allocation relationships. Analytical support for performance assessment (i.e., simulation) and trade studies is handled through API (application programming interfaces) linkages to engineering analysis tools such as MATLAB and Modelica [42, 43].

In recent years the trend toward performance-based design and operation of systems has elevated the importance of requirements maintenance and management. Unlike past generations of

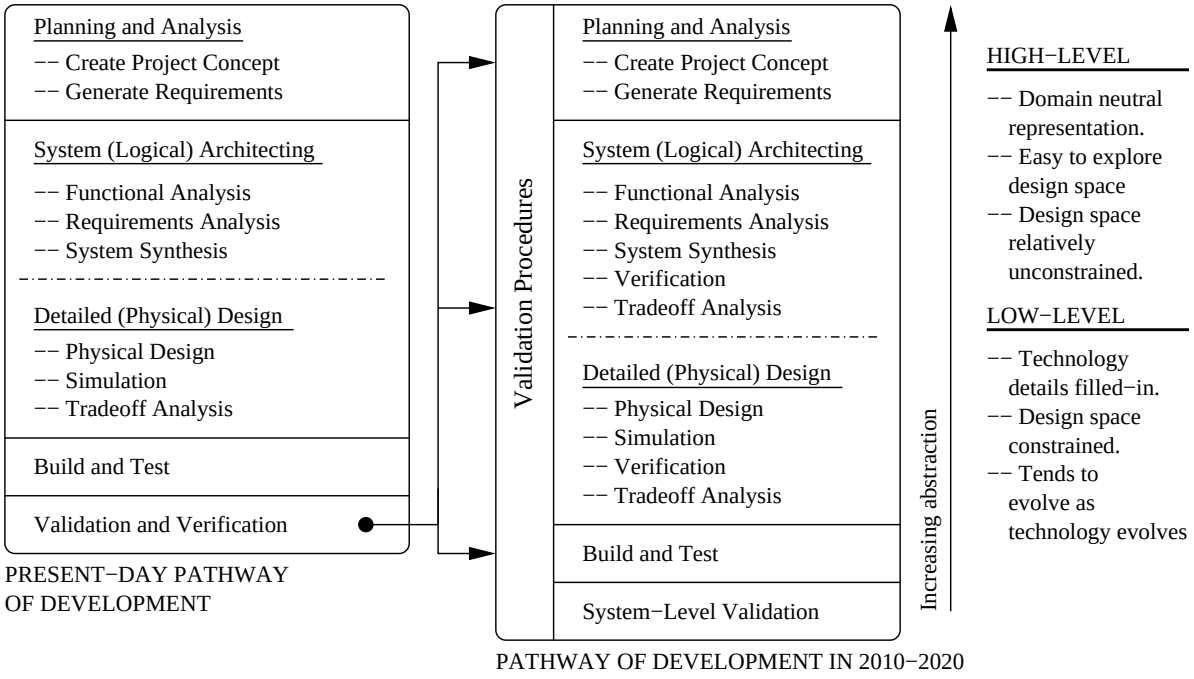



Figure 1.4: Pathways of development, now and in 2010-2020

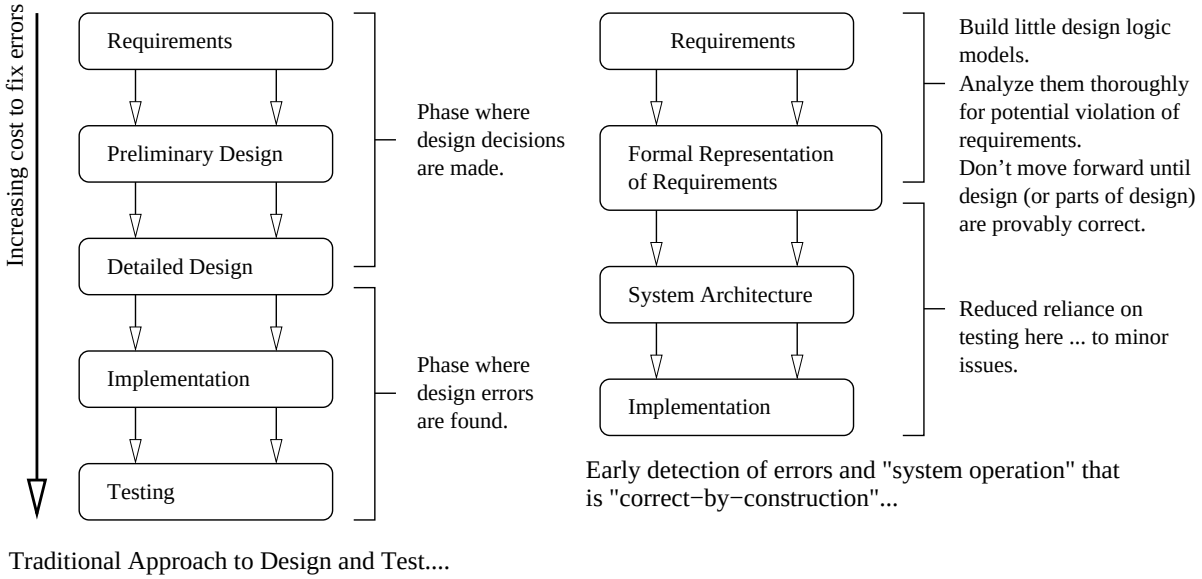



Figure 1.5: Pathways of traditional and model-based system development (Adapted from Sidorova [55]).

system development, these systems are required to have levels of performance that satisfy requirements throughout the entire lifecycle of a system. One implication of this change lies in the modeling of systems. While it may have once been acceptable to implement requirements management on system representations that were rather abstract (see, for example, the description of SLATE in Chapter 2), requirements management systems now need to work with a variety of engineering abstractions, including representations of the final design. Looking ahead, the required changes can only benefit systems engineering. Recent history tells us that the benefits of UML/SysML 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 [21, 22]. These gaps will not be bridged unless a method is found to use UML and SysML (and their inevitable extensions) in concert with discipline-specific models and notations (e.g., visualization of requirements; block diagrams; two- and three-dimensional engineering schematics).

## 1.2 Objectives and Scope

The long-term objectives of this research are to mitigate these shortcomings, thereby providing a pathway for the computer to play a pro-active role in the synthesis and formal checking of multi-disciplinary system architectures. In a departure from other development efforts, the underlying tenet of our research is that end-to-end development of engineering systems will occur through multiple models of computation, control, and visualization networked together. We assume that computational implementations will correspond to 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 associated graphical constructs will promote a shared comprehension of relationships between disciplines, and patterns of change and negotiation (particularly, cause-and-effect and trade-off of functionality, performance and cost) within collaboratively developed systems. Traceability mechanisms are the glue that will bind multiple models of engineering development and visualization together. Present-day systems engineering methodologies and tools are not designed to handle projects in this way.

In this report we take a first step toward the implementation of this vision through development of a new methodology and system for ontology-enabled traceability. By making ontologies

an integral part of traceability mechanisms we hope to achieve several objectives:

1. Ontologies carry with them a conceptual representation and understanding of a particular domain. By explicitly connecting requirements to engineering system representations through ontologies we are indicating "how and why" requirements satisfaction is taking place.

2. Ontologies for different design viewpoints (e.g., system structure, system behavior) may also be linked, thereby establishing dependencies among the viewpoints of different engineering disciplines and their concerns.

3. Part of the understanding of ontologies is rules that partition acceptable functionality and performance from unacceptable functionality and performance. A third benefit of the proposed method is the opportunity for design rule checking at the earliest possible moment in design. This is where design errors are easiest and cheapest to fix. For an operational system that is being monitored, real-time evaluation rules can also contribute to system management.

Chapter 2 covers state-of-the-art requirements modeling and traceability, and describes in detail, traceability capabilities of the IBM Teamcenter (SLATE) Requirements Tool; it is arguably best-of-bred in traceability capability. Details of the proposed traceability model and its relationship to ontologies and ontology-enabled computing are presented in Chapter 3. The software architecture design and details of implementation (e.g., mechanisms of event-based communication) are described in Chapter 4. Our preliminary implementation is a requirements-to-ontology-to-application software prototype for a simplified representation of the Washington DC Metro System. A description of this pilot application may be found in Chapter 5. Finally, background material on inspiration for the proposed model and semantic web technologies is presented in Appendices A.1 and A.2.

# Chapter 2

# State-of-the-Art Modeling, Traceability, and Visualization of Requirements

The purpose of this chapter is to critically examine state-of-the-art modeling, traceability, and visualization of requirements.

Real-world engineering systems are developed over multiple levels of abstraction (i.e., system, subsystem, component levels) using pre-defined strategies of development that are part top-down decomposition and part bottom-up assembly. Throughout the development process, teams need to maintain a shared view of the project objectives (this implies good communication among stakeholders and developers), and at the same time, focus on specific tasks. To ensure that the development process moves forward in a disciplined manner, pre-defined processes are needed for requirements development (elicitation, organization, visualization), system synthesis and design, integration and validation. Two key elements of this capability are an ability to identify and manage requirements during all phases of the system design and operational lifecycle.

## 2.1 Pathway from Operations Concept to Requirements and System Design

To see how these principles apply in practice, let us assume that the required engineering system does not exist. Figure 2.1 illustrates the development pathway for one level of abstraction,

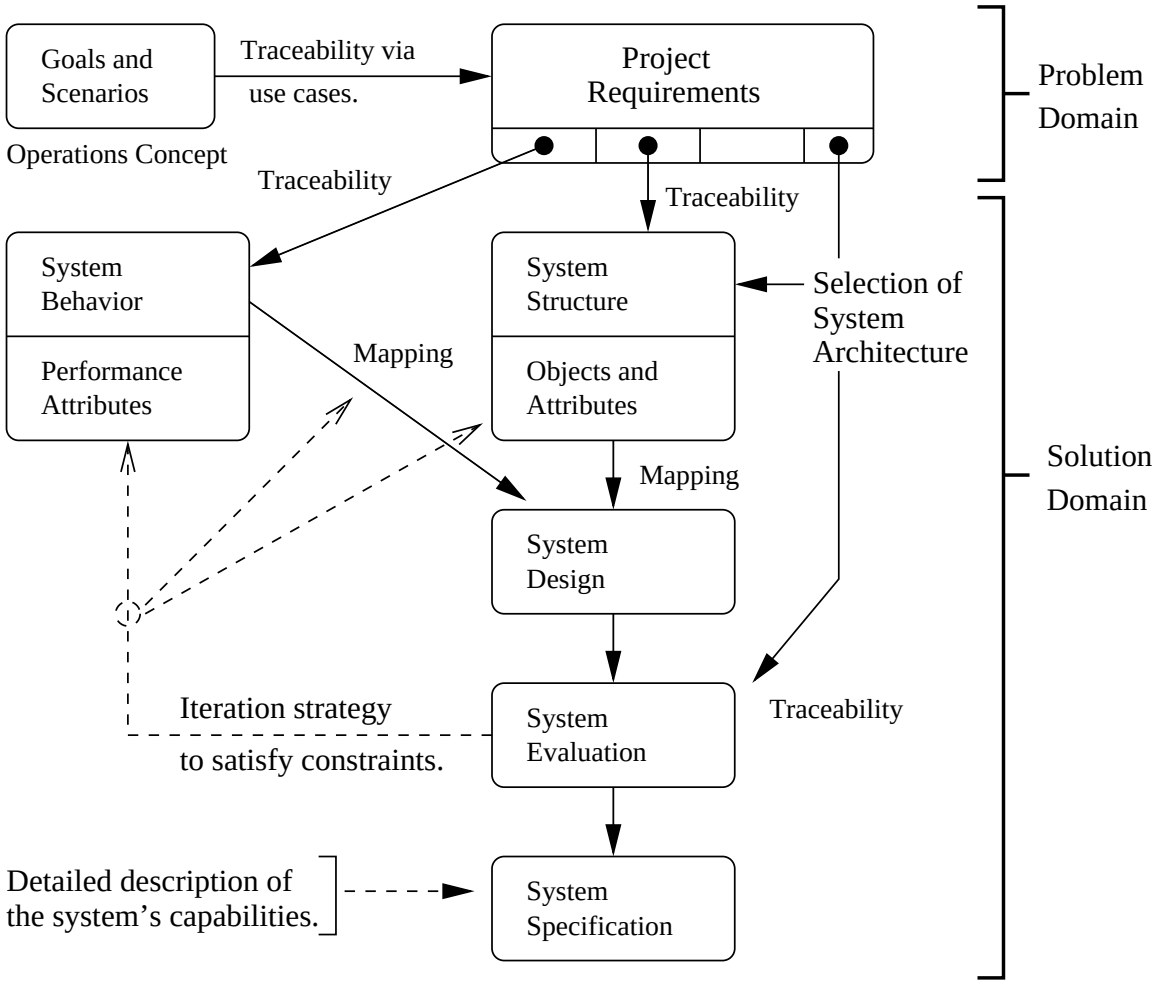



Figure 2.1: System structure and system behavior viewpoints, and traceability mappings for the development pathway goals/scenarios through system evaluation.

beginning with the formulation of an operations concept, requirements, fragments of behavior, and tentative models of system structure. Requirements need to be organized according to role they will play in the design (e.g., behavior, structure, test) and processed to insure consistency, completeness, and compatibility with the requirements system. Models of behavior state what the system will do. System performance can be evaluated with respect to the value of performance attributes. Models of structure specify how the system will accomplish its purpose. System architecture will be evaluated with respect to selected objects, and the value of their attributes. System designs are created by assigning (or mapping) fragments of required to object and subsystems in the system structure. Thus, the behavior-to-structure mapping defines (in a symbolic way) the functional responsibility of each subsystem/component. Finally, in the system evaluation, functional and performance characteristics are evaluated against the test requirements. To satisfy all of the system requirements, several iterations of development (involving modifications to the operations concepts, system behavior, system structure) will usually be required.

## 2.2 Low- and High-End Traceability

Now that documents containing thousands and, sometimes, tens-of-thousands of requirements are commonplace, requirements modeling and traceability management tools are an indispensable enabler of the system development process. Traceability mechanisms allow for an understanding of how and why various parts of the system development process are connected, thereby providing the development team greater confidence in: (1) Meeting objectives; (2) Assessing the impact of change; (3) Tracking progress; and (4) Conducting trade-off analysis of cost against other measures of effectiveness. Visualization mechanisms improve the effectiveness in which engineers can understand the problem under development.

In a comprehensive study of traceability models and meta-models, and their use in industry, Balasubramaniam and co-workers [5] have classified users of traceability into two categories. Low-end users have problems that require less than about 1,000 requirements (viewed as a mandate from the project sponsors or for compliance with standards). They typically view traceability as a transformation of requirements documents to design; they also lack lack support for capturing rationale for requirements issues and how they are resolved.

High-end users of traceability tend to have problems that require, on average, about 10,000

requirements (viewed as a major opportunity for customer satisfaction and knowledge creation throughout the system lifecycle) [5]. They view traceability as an opportunity to increase the probability of producing a system that meets all customer requirements, is easier to maintain, and can be produced within cost and on schedule. High-end traceability employs much richer schemes of traceability (e.g., capture of discussion issues, decisions and rationale product-related and process-related dimensions) than their low-end counterparts. Traceability pathways of rationale enable accountability (e.g., what changes have been made; why and how they were made), particularly to stakeholders not directly involved in creation of the requirement.

## 2.3 State-of-the-Art Capability

This section contains a critical assessment of state-of-the-art capability in: (1) Requirements modeling and traceability, and (2) Visualization of requirements, as implemented in modern-day requirements management tools.

### 2.3.1 Part I. Requirements Modeling and Traceability

Present-day requirements management tools such as SLATE [30], CORE [12], and DOORS [17] provide the best support for top-down development where the focus is on requirements representation, traceability, and allocation of requirements to system abstractions. In most of today's requirements management tools, individual requirements are represented as textual descriptions with no underlying semantics. System engineers like to organize groups of requirements (e.g., functional requirements, interface requirements) and abstractions for system development into tree-like hierarchies, in part, because this technique is comfortable and well known. This is state-of-the-art practice. However, when requirements are organized into layers for team development, graph structures are needed to describe the comply and define relationships, sometimes tracing across the same level. This happens because requirements are tightly interdependent with each other across the same level of abstraction. Because the underlying graphical formalism is weak, many questions that a user might want to ask about requirements and/or the system structure remain unanswered or omitted. Simple questions like "Show me all complying and defining requirements that are related to this particular requirement" cannot be answered.

As a case in point, the IBM Teamcenter (SLATE) Requirements Tool aims to improve

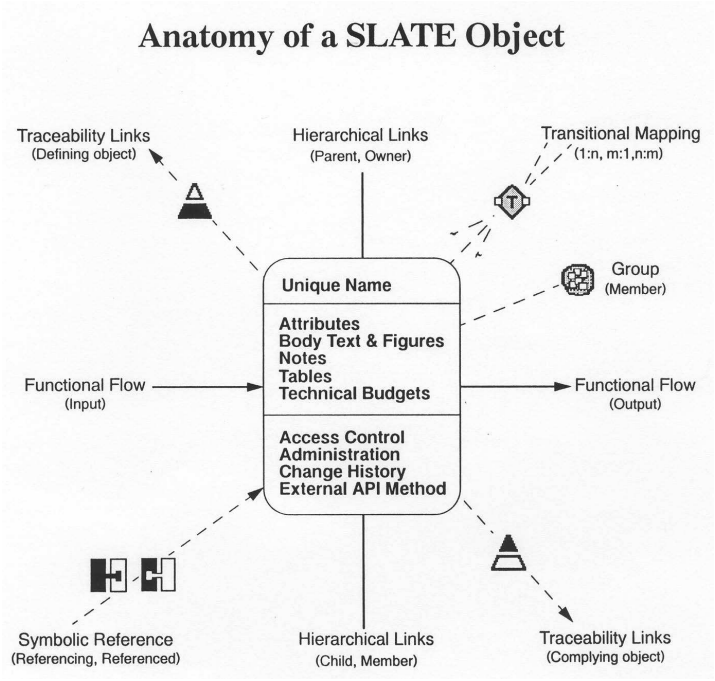



Figure 2.2: Anatomy of a generic object in IBM Telelogic SLATE [30].

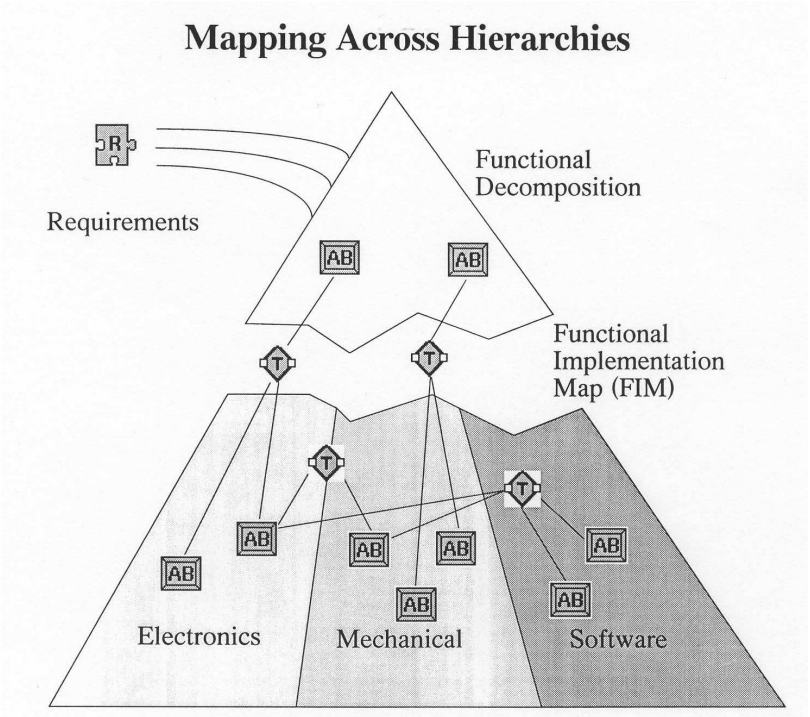



Figure 2.3: Modeling of translational mappings (TRAMs) across hierarchies in IBM Telelogic SLATE [30].

systems engineering productivity (e.g., better accuracy; accelerated functional design; support for trade studies), particularly at the conceptual stages of development. SLATE is based upon very good data representations for requirements and abstraction blocks (ABs), linked together into a graph structure. The graph edges correspond to relationships between entities in the system development – for example, traceability links (complying and defining links) and connectivity of different abstraction block hierarchies with translational mappings. ABs provide modeling support for attributes (whose values can be used in performance assessment), functional flows (i.e., data inputs and outputs), links to other ABs (e.g., in parent/child relationships), connectivity to groups and translational mappings, and budgets. See Figure 2.2. Justification for use of abstraction blocks in lieu of more detailed representations of an engineering system is really very simple – at the conceptual stages of development, most of these details (e.g., geometry) remain to be developed.

Translational mapping relationships (TRAMs) provide a method for connecting abstraction blocks across hierarchies and for evaluating design alternatives. The upper part of Figure 2.3 shows, for example, trace links connecting requirements to abstraction blocks in a functional decomposition hierarchy. Then, TRAMs relay the existence of dependencies between ABs in the electrical, mechanical and software viewpoints. Translational mappings (TRAMs) work in terms of connecting source ABs to destination ABs, and source-to-destination and destination-to-source pathways. Two examples are shown in Figures 2.4 and 2.5.

### 2.3.2 Part II. Visualization of Requirements

Effective visualization techniques help end-users understand and study the behavior and underlying cause-and-effect mechanisms within a phenomena [25]. Unfortunately state-of-the-art capability in requirements visualization falls short of these goals and, in fact, has not advanced much during the past two decades. Prior to 2006 requirements visualization has been used primarily for three purposes: (1) To convey the structure and relations among evolving requirements and other system artifacts, (2) To support the organization of requirements and, downstream, the management of requirements during change, and (3) To model subsets of requirements (or properties of these requirements) for analytical/engineering purposes. Looking forward, one can imagine requirements visualization techniques mapping data/information about requirements onto visual artifacts, permitting designers to "actually see" the requirements in the context of their satisfaction and support for high-level decision-making activities.

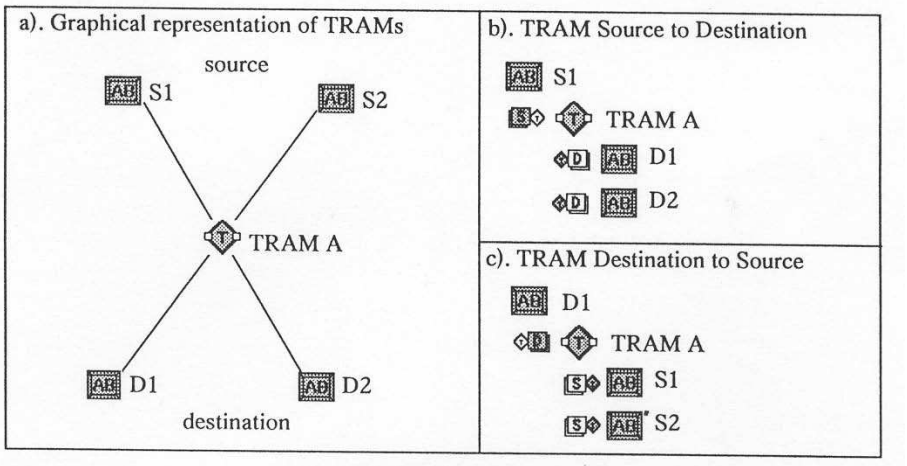



Figure 2.4: Modeling and graphical representation of translational mappings (TRAMs) in SLATE. Part a shows a scenario with two source abstraction blocks and two destination abstraction blocks. Part b shows an outline view of the icons that will be displayed when the TRAM is expanded from one of the source source abstraction blocks. Part c shows the icons that will be viewed when the TRAM is expanded about one of the destination abstraction blocks.

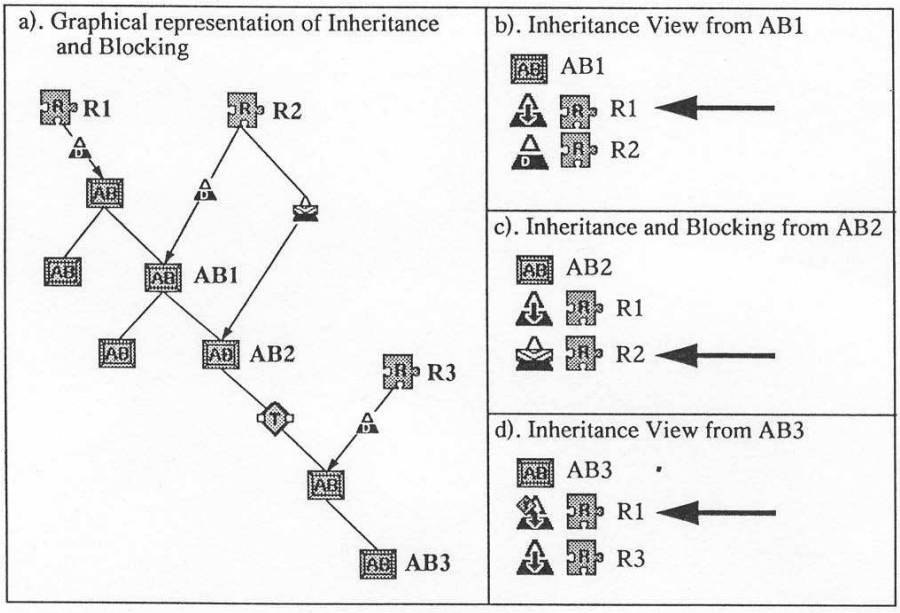
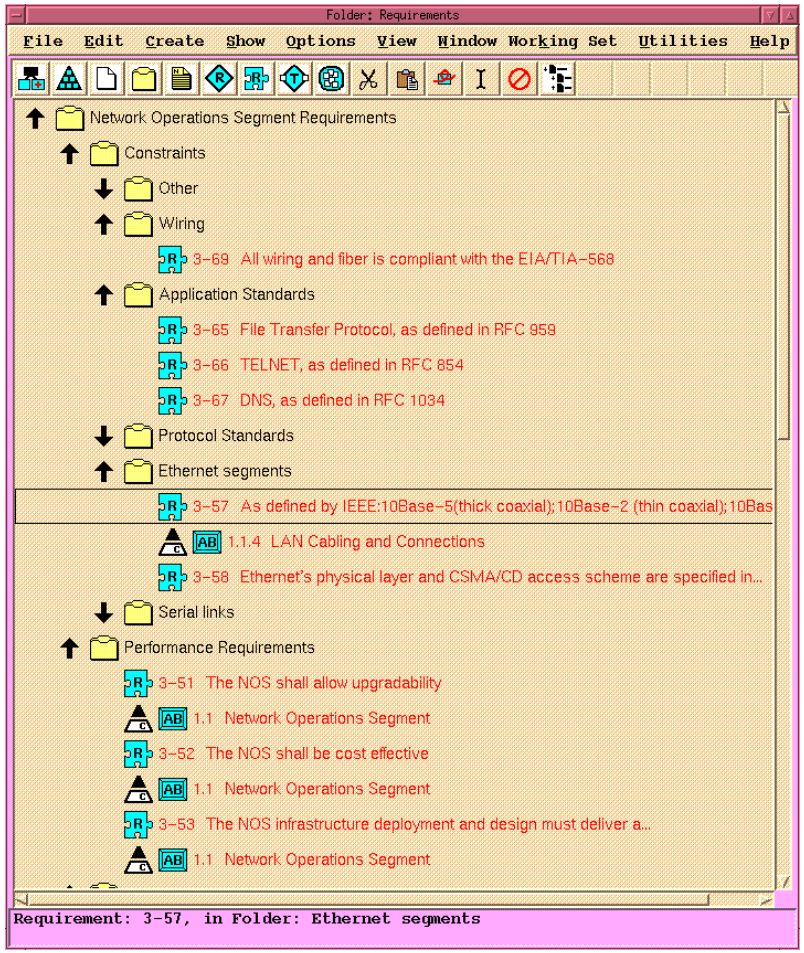
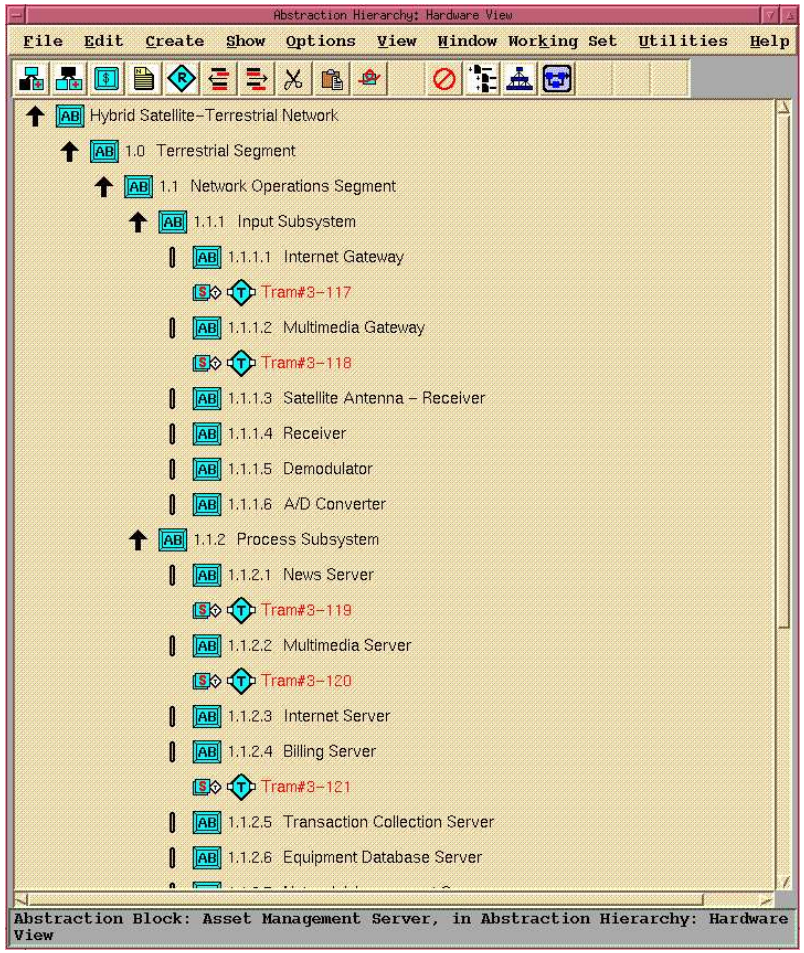



Figure 2.5: Inheritance and blocking Mechanisms in SLATE. Three cases: (1) Abstraction block 1 complies with requirements R1 and R2, (2) Abstraction block 2 complies with requirement R1, but not R2, (3) Abstraction block 3 is defined by R3 – It also complies with R1 through the TRAM mechanism.




(a) Folder visualization of requirements in SLATE.




(b) Representation of abstraction block hierarchy and translational mappings (TRAMs) in SLATE.

To put these comments on a sound footing let's return to SLATE, where system-level designs are viewed as collections (e.g., networks and hirarchies) of functional units (ABs) that form the major components of a system. The outline viewpoint highlights dependencies among ABs. The block diagram viewpoint focuses on data flows between ABs. Unfortunately, the underlying graphical support is weak in the sense that no provision exists for viewing a more detailed representation of the system after lower-level details have been worked out. For systems that require monitoring throughout their working lifetime, this is a major deficiency. 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 2.3. Instead, requirements and abstraction block hierarchies with translational mappings among (TRAMs) viewpoints are visualized using the notation shown in Figures 2.6(a) and 2.6(b).

**Summary**

Together these weaknesses leave systems and 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.

# Chapter 3

# Ontology-Enabled Traceability

In a first step toward mitigating the weaknesses in state-of-the-art capability in requirements modeling and traceability, in this chapter we propose a new approach to requirements traceability. We begin with a description of the simplest model possible – one requirement is satisfied by applying a single design concept which, in turn, is implemented with a single design object. Then, we propose extensions to the traceability model where real-world systems are designed from multi-functional components and to satisfy the needs of multiple stakeholders.

## 3.1 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 or domain [24, 27, 56]. To provide for a formal conceptualization within a particular domain, and for people and computers to share, exchange, and translate information within a domain of discourse, an ontology needs to accomplish three things [37]:

**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. This project assumes that advances in ontology-enabled design and development will occur in parallel with advances in the Semantic Web; for details see Appendix A.2.

The ontology community makes a distinction between ontologies that are taxonomies and those that model domains in depth, applying restrictions on domain semantics [24]. 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.

## 3.2 Proposed Approach to Traceability

In a first step toward mitigating the weaknesses in state-of-the-art capability in requirements modeling and traceability, in this chapter we propose a new approach to requirements traceability. Figure 3.1 provides a simplified view of state-of-the-art traceability and the proposed model.

The upper half of Figure 3.1 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). Physical embodiments are the most natural interpretation of design/engineering objects; however, the design itself may be a conceptual non-physical system. Looking forward, 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. Under design occurs when requirements cannot be traced forward to the design (i.e., they have not been taken into account). Over design occurs when the design contains objects and systems that cannot be traced back to a requirement (i.e., the design contains features that are not needed).

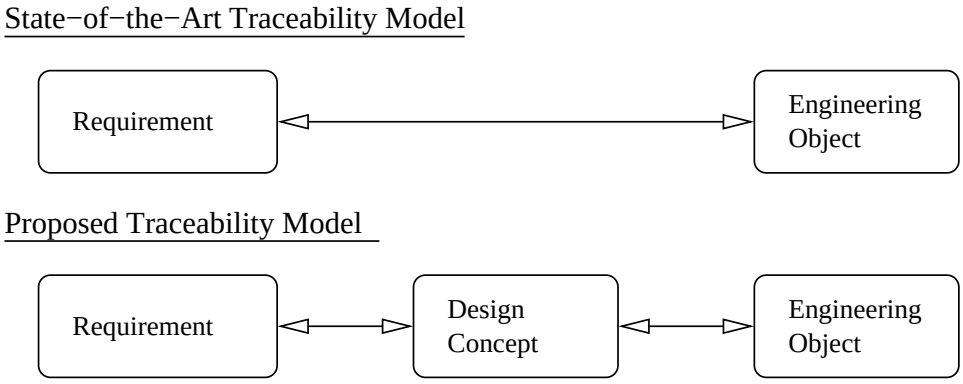



Figure 3.1: Simplified view of state-of-the-art traceability and the proposed model.

The lower half of Figure 3.1 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 which, in turn, will be evaluated in terms of the values of attributes relevant to the concept. Thus, the links between design concepts and engineering objects represents an actual implementation of concepts.

**Benefits.** The proposed method offers the following benefits:

1. From an efficiency standpoint, the use of ontologies within traceability relationships helps engineers deal with issues of system complexity by raising the level of abstraction within which systems may be represented and reasoned with. Furthermore, because ontologies represent concepts for a problem domain, the ontologies are inherently reusable.

2. 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.

3. 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).

**Remark.** Inspiration for the proposed approach to traceability stems from the domain of architectural design for modern building environments. As a focus for study this problem domain is appealing because it is easy to understand, yet, good solutions demand a team-based approach to development with input and coordination of activities from multiple disciplines. A detailed discussion of the issues can be found in Appendix A.1.

## 3.3 Representing Design Concepts 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 A.4.

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 [14, 13]. 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 [3]. 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. [35].

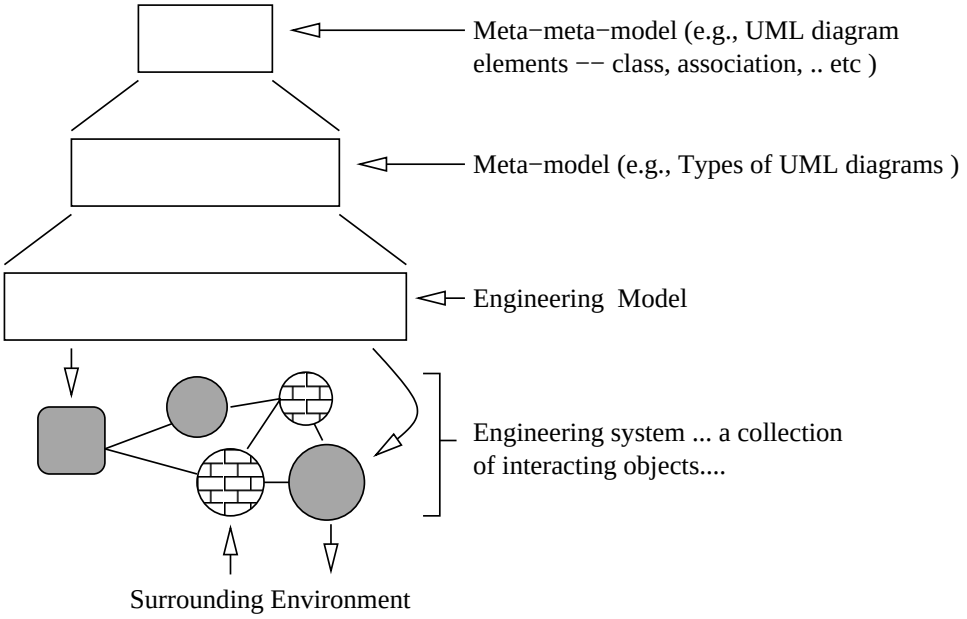



Figure 3.2: Pathway from Meta-meta-models to Engineering Models and Systems (Source [65])

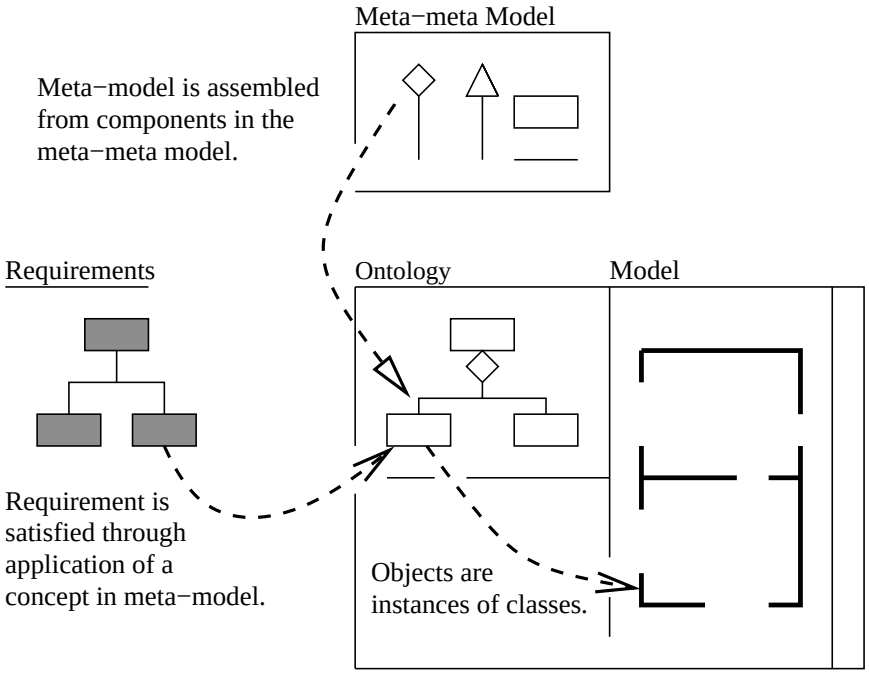



Figure 3.3: Roles of the Meta-Meta-Model and Meta-Model in the System Assembly

**Meta-Model for the Proposed Approach.** 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. And meta-meta-models describe information about meta-models (i.e., a language in which meta-models may be expressed). Figure 3.2 shows the pathway from meta-meta-models to meta-models to UML models and implementation of engineering systems. Key points to note are as follows:

**1.** The meta-meta-model (also known as the UML meta-model) is a model that describes concepts in 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 (i.e., rules and meaning) 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 lments 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.** UML 2 is defined by eight diagram types for behavior and six diagram types for system structure.

In established approaches to engineering/software design, UML diagrams are created for required system behavior (e.g., activity and statechart diagrams), system structure (e.g., class and object diagrams), and the mapping or assignment of required behavior to system structure (e.g., collaboration diagrams). The relationship among diagram elements (e.g., an association between two classes in a class diagram) can imply a functional relationship (e.g., connectivity, required sequencing of activities) that must exist in the design. Most often, these relationships are statements of required functionality. Performance and interface requirements will stem directly from these diagrams.

Figure 3.3 shows the role of UML diagram elements and specifically UML class diagrams in the proposed approach. In a departure from established approaches to design, our goal is to represent and visualize engineering models in a manner that consistent with discipline-specific notations. For example, an architect might use lines to visualize the simple geometry of a house. We envision that requirements will be organized into layers of detail suitable for team-based development (although it is conceivable that the might also be represented via SysML). 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 3.3.

## 3.4 Frameworks for Multiple-Viewpoint Design

The lower half of Figure 3.1 is overly simplified in the sense that it implies one requirement can be satisfied by the application of one design concept or function which, in turn, will lead to the implementation of one engineering object. Real-world systems are much more likely to correspond to assemblies of design entities, organized into hierarchies along disciplinary lines, with each design entity representing a meaningful concept to one or more system stakeholders [49]. To accommodate these relationships in a disciplined way, there needs to be a formal framework for: (1) Connecting stakeholder concerns to engineering entities, (2) Capturing the interactions and restrictions among the various viewpoints, and (3) Systematically abstracting away details of a problem specification that are unrelated to a particular decision.

### 3.4.1 Frameworks for Modeling Architectural Descriptions

As a first step toward understanding how stakeholder perspectives and needs might be integrated into a comprehensive system model, Figure 3.4 shows the essential concepts and relationships among concepts involved in the development of architecture descriptions.

The principal concepts are as follows:

1. An architecture is a fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

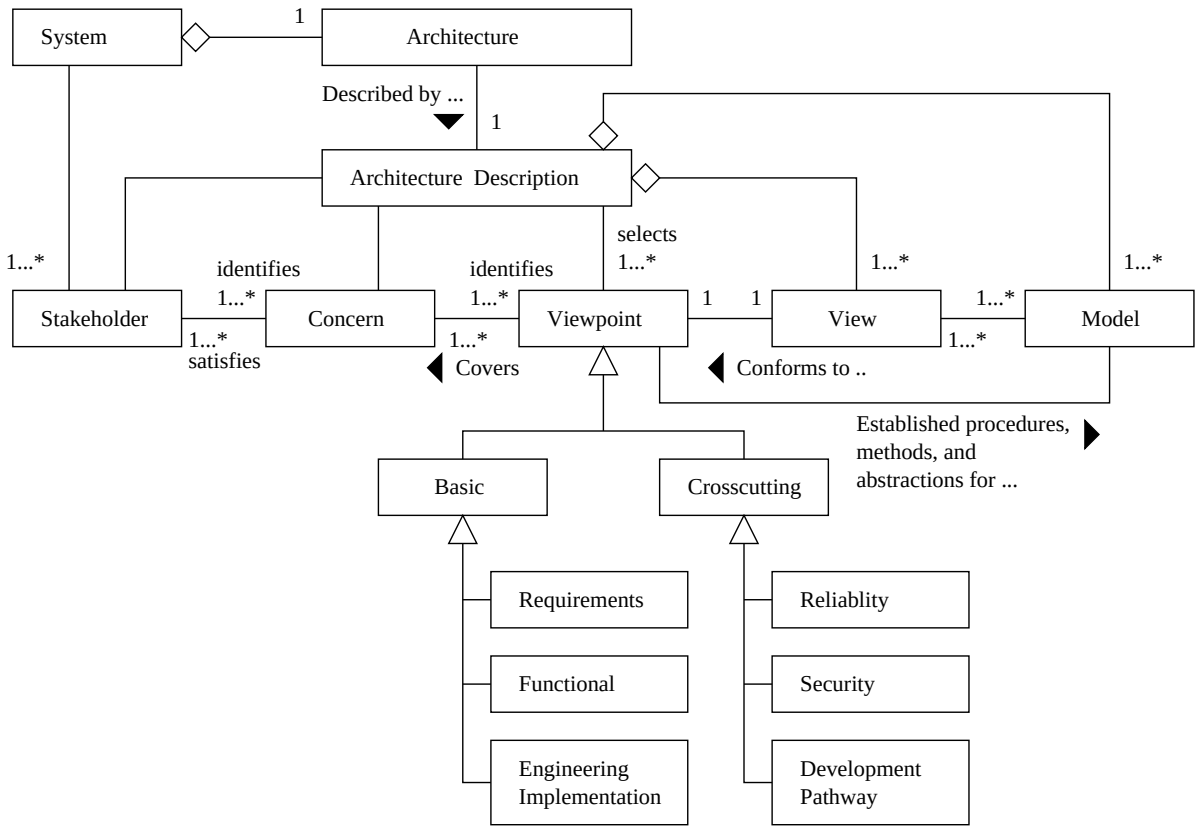



Figure 3.4: Class diagram of concepts contributing to development of architecture descriptions. Assembled from ideas due to Eeles et al. [19], Maier [40], and definitions in the IEEE 1471 Standard [31].

**2.** A system stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.

**3.** Concerns are those interests which pertain to the systems development, its operation, or any other aspects that are of critical importance to one or more stakeholders. Typical concerns include considerations such as system functionality, performance, reliability, security, distribution, ease of evolvability, schedule of development, and maintenance and cost.

**4.** A view is a representation of a whole system from the perspective of a related set of concerns.

**5.** A viewpoint is a specification of the conventions for constructing and using a view (i.e., notice that there is a one-to-one correspondence between a view and a viewpoint). As such, the viewpoint determines the languages (including notations, model, or product types) that will be used to assemble the view, as well as any associated modeling/analysis techniques.

These languages and techniques are used to yield results relevant to the concerns addressed by the viewpoint. For example, the class and statechart diagram types in UML define the semantics for representing diagrams that aid engineers in understanding system structure and behavior, respectively. A second example of this process is the multi-resolution capabilities of Google Maps. When multiple visual representations of the same model are needed (e.g., different projection views of a house), software implementations should follow the model-view-controller (MVC) design pattern.

**6.** Viewpoints may be partitioned into basic viewpoints and crosscutting viewpoints.

Basic viewpoints are associated with views that can be represented by a singular type of model or entity (e.g., a requirements model, a functional model, a specific module or subsystem). Crosscutting viewpoints cut across basic viewpoints, for example, multiple stages of development (e.g., requirements, implementation) and/or multiple subsystems (e.g., to evaluate system reliability and/or security).

**7.** Architectural models are developed using the procedures and methods established by the associated architectural viewpoint.

The march toward enhanced functionality and higher performance means that, increasingly, systems entities are required to be multi-functional, which, in turn, means that they will participate in the satisfaction of multiple stakeholder needs and their associated viewpoints. Hence, there is a

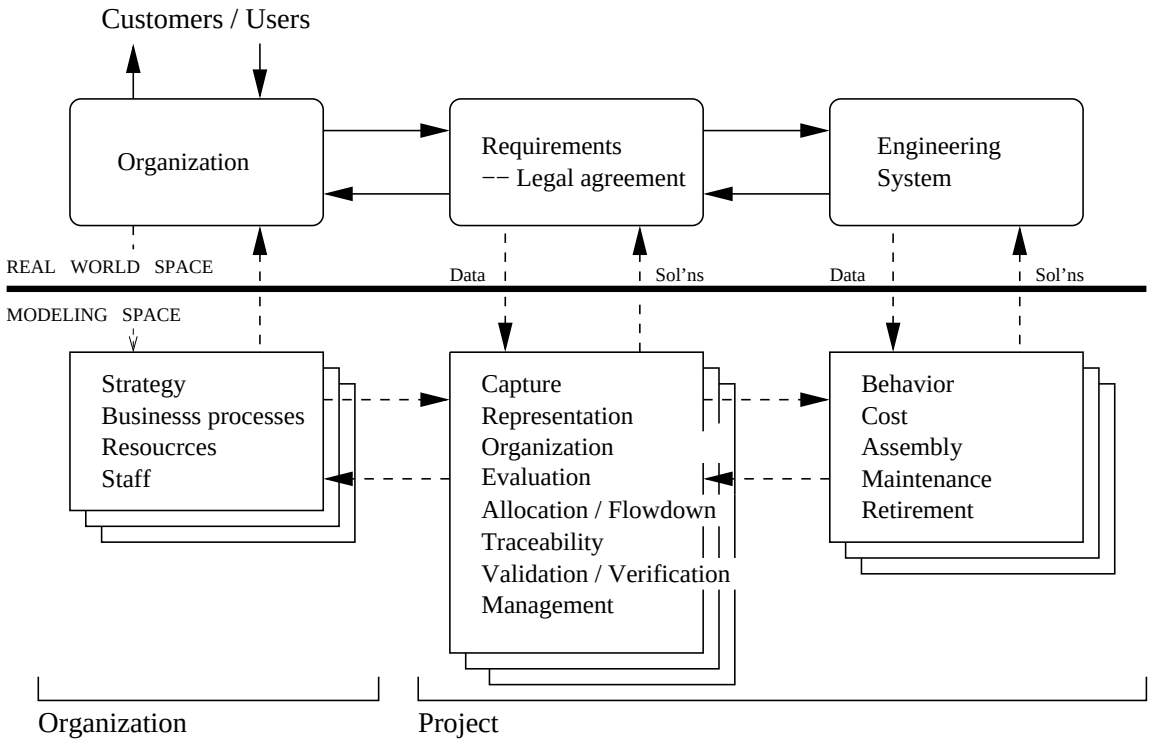



Figure 3.5: Requirements bridge the gap between organization- and project-level development.

strong need to represent not only multiple perspectives in design, but relationships between these perspectives. Overlaps in system functionality must be identified. Complementary participants must be made to interact and cooperate. Contradictions must be resolved.

Systems are developed to satisfy the needs of one or more stakeholders. Each stakeholder typically has interests in, or concerns relative to, that system. The uppermost layer of this diagram says that systems have an architecture which, in turn, is described by one architecture description. The architectural description is organized by one or more views and one or more architectural models. Then, in turn, an architectural description selects one or more viewpoints for use. Each view addresses one or more of the concerns of the system stakeholders. Reading left to right along the bottom of the figure, stakeholders have one or more concerns, which are covered by viewpoints, views and models. Reading from right to left, an architectural model may participate in more than one view, each conforming to a viewpoint developed to answer questions about specific stakeholder concerns. Figure 3.5 shows, for example, elements of organization- and project-level development and the associated engineering, requirements and engineering design concerns.

**Remark.** Figure 3.4 is simply a snapshot of multiple-viewpoint design and is readily extensible. For example, the IEEE 1471 Standard contains relationships that link a system to the goals of a mission and the underlying rationale driving the synthesis of an architectural description.

### 3.4.2 Mechanisms for Functional and Viewpoint Interaction

Figure 3.4 sets the stage for multiple viewpoint design and asserts that architecture descriptions are inherently multiview, with no single view adequately capturing all of the stakeholder concerns. The potential complexity of this problem is indicated by the chain of many-to-many relationships between stakeholders and their concerns, and then concerns and their study through the implementation of multiple viewpoints. The first limitation of Figure 3.4 is that it does not explicitly describe how the concerns associated with the various viewpoints will actually interact. In some cases the relationship between concerns will be purely symbolic (e.g., entities A and B are the same). But dependencies might also be physical, requiring an understanding of notions such as connectivity and constraint, flows of data/energy, and scheduling and coordination. A second limitation of Figure 3.4 is that it does not deal with the issue for how the models of an object/system will relate to the actual physical object/system. To overcome these problems, this section deals with these issues by looking at functional and viewpoint interaction at two levels of abstraction: (1) the model level, and (2) the meta-model level.

The nature of dependency and interaction relationships is not as simple as one might initially think. Figure 3.6 shows, for example, a system organized into (disciplinary-specific) hierarchies and three examples of dependency relationships between viewpoints: (1) same as (i.e., the element has all of the properties of the "named" element), (2) element of (i.e., the element is a component of the "named" element), (3) part of (i.e., the element forms part of the "named" element), and constrained by (i.e., a property of an element is constrained by the property of another). Figure 3.6 implies that design entities will be viewed in a consistent manner. However, a much more common situation is that each discipline will model and view design objects relevant to their set of concerns, and as indicated in Figure 3.7, may not even use the same terms to describe the same design object. For example, building architects refer to horizontal planes as floors. Structural engineers refer to the same object as a slab. And, in fact, because these disciplines often work at different stages of project development, neither single unified object models, nor single unified system models can be guaranteed to exist.

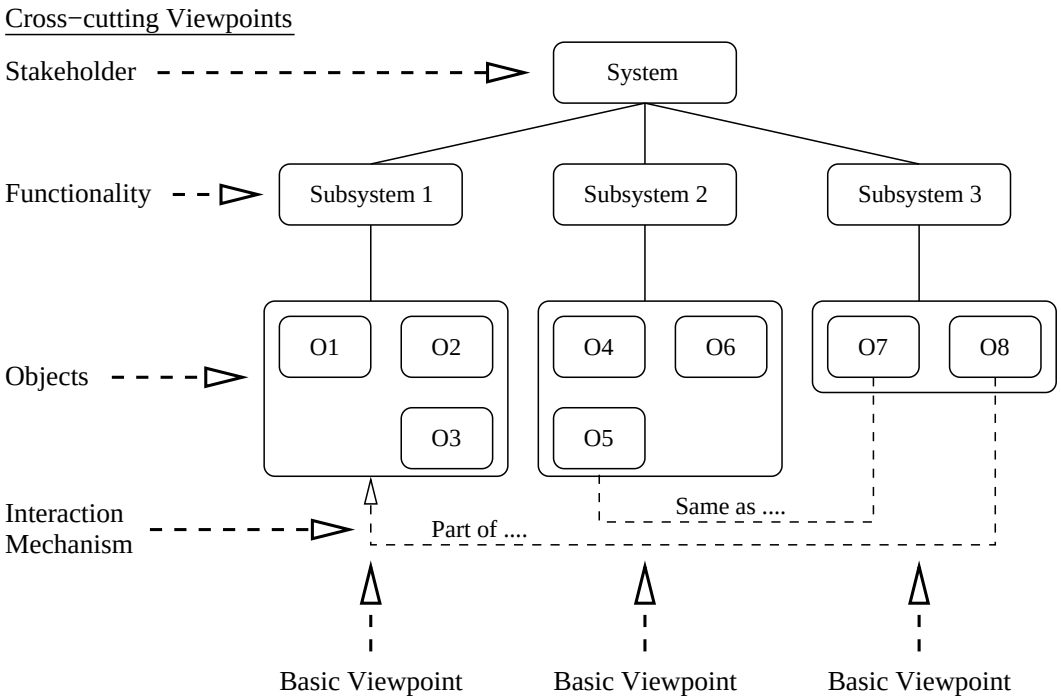



Figure 3.6: Example of interaction mechanisms connecting viewpoints associated with system decomposition.

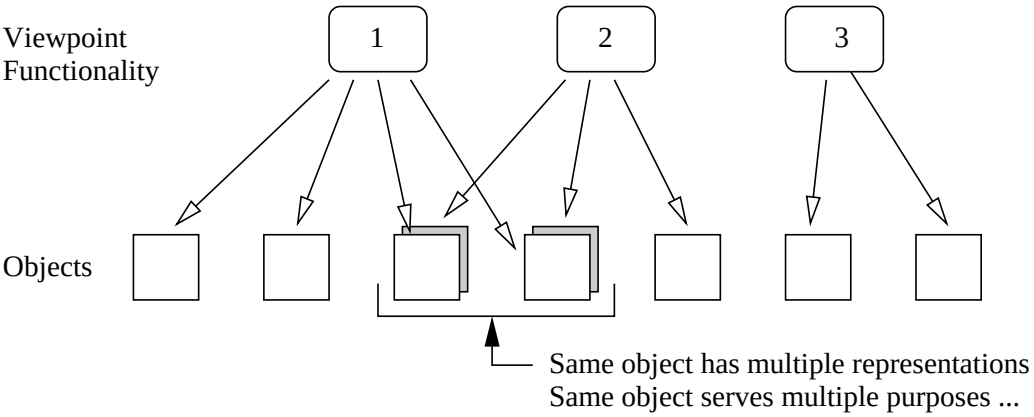



Figure 3.7: Multiple conceptual interpretations of a single design object. Because most systems are developed in stages, a single unifying model of an object and/or the system may not exist.

**Functional and Viewpoint Interaction at the Model Level.** During that past two decades, several models of viewpoint interaction have been proposed.

In the early 1990s, systems engineers at TD Technologies (now owned by IBM) implemented translational mappings in SLATE as a means of linking: (1) functional hierarchy abstractions to abstraction block hierarchies in a variety of disciplines (e.g., mechanical, electrical, software), and (2) dependencies among disciplines. The basic model for domain interactions and propagation of dependencies is translational mappings, as illustrated in Figure 2.3. Figures 2.4 and 2.5 show graphical notations for source-to-destination pathways (e.g., an abstraction block complies with a requirement) and the linkage of requirements and abstraction block hierarchies and satisfaction relationships through the use of translational mappings. TRAMs are weak in the sense that they acknowledge the existence and direction of a relationship between abstraction blocks, but otherwise leave it up to the designer to determine the meaning and context of the relationship.

In the mid 1990s, several research teams [48, 49] developed methodologies to handle multi-functional design concerns through the systematic identification of attributes associated with required functionality, followed by their organization into groups called primitives. This process is illustrated in Figures 3.8 and 3.9. An object primitive is defined to be a group of closely related object attributes that provide a design-concern-focused abstraction to the designer. For example, attributes might be clustered according to dimensions (geometric concerns), thermal properties (energy usage concerns), structural properties and loadings (structural design concerns) and cost (economic concerns). Then, working backwards, primitives are used to present the different views of a system entity. Views hide the actual complexity of the entity by providing only the essential information needed for decision making relating to a specific design concern, and systematically removing all of the other details.

**Functional and Viewpoint Interaction at the Meta-Model Level.** The key benefit in supporting dependencies among viewpoints at the system entity level is that it provides a complete picture of all of the participating viewpoints and functions associated with the entity. However, this approach always starts from scratch and requires a top-down decomposition of the system into entities that can be characterized by design-concern-focused abstractions. No built-in support is provided for reuse across families of similar projects. In practice, a global vision for how a specific project's goals will be satisfied (top-down refinement of abstractions) needs to be balanced against a bottom-up synthesis of context-specific abstractions and practices (e.g., models of cooperation,

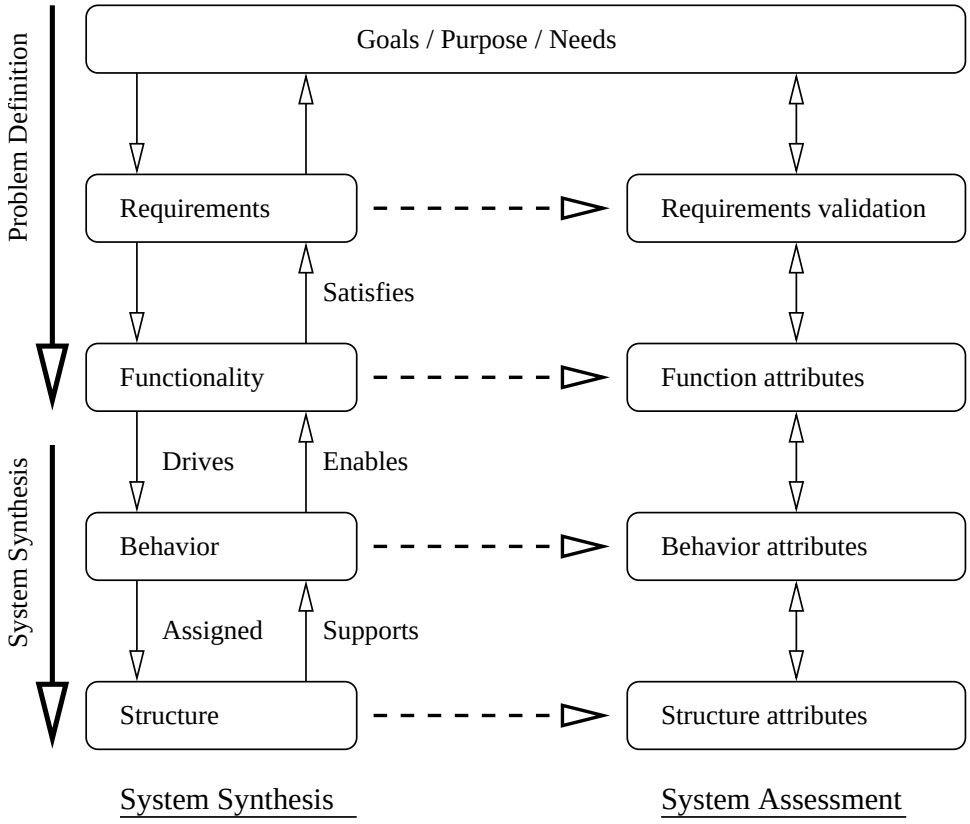



Figure 3.8: Synthesis of attributes for multiple-viewpoint design.

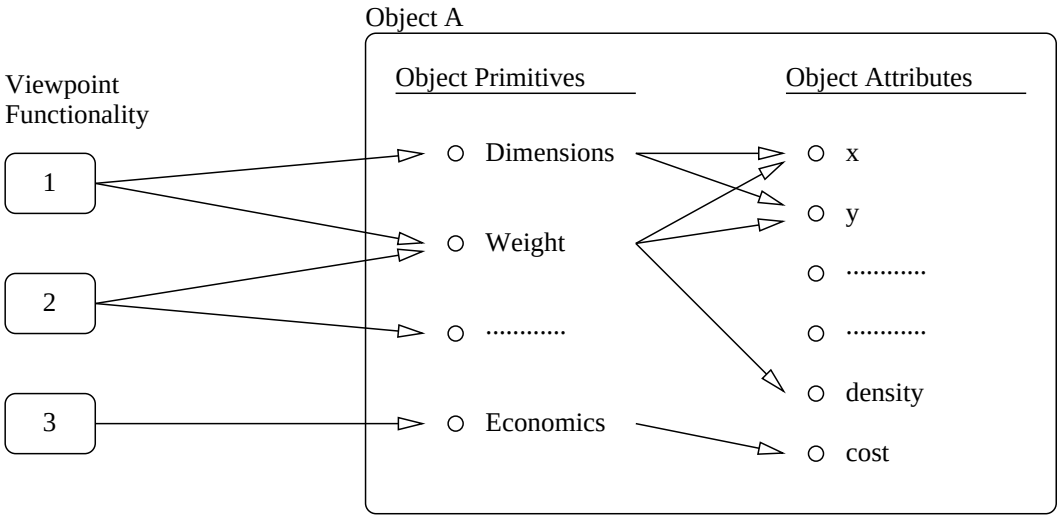



Figure 3.9: Framework for modeling multiple views of a single design object. Design concerns associated with viewpoint functionalities are linked through a common primitive model.

exchange, dependencies among the project participants) that are repeated across families of similar projects.

During the past decade the Object Management Group has developed an approach called model driven architecture (MDA) for software systems development and, in particular, a framework of certified industrial standards (e.g., UML, MOF). The MOF structure [45] gives us a conceptual framework to represent collective activity and knowledge relations among collaborative entities. The goals of model-driven engineering (MDE) are to unify different technical spaces (XML, Ontology) and, as such, recommend the use of meta-models to define domain languages. After an abstract layer has been defined in the context of collaboration – that is, the relationships among elements in a project – in various different domains, specific models are created through instantiation of domain-specific models. One can then create design tool support for customized models and viewpoints relevant to an end-users specific needs, thereby helping them to monitor and understand a system's state and possibly look ahead and anticipate future states of the system.

The benefits of functional and viewpoint interaction at the meta-model level can be found in cooperative platforms for building construction [26], where it is well known that during the assembly stages of system development, final quality depends highly on the cooperation between actors (or teams of actors) which may not have a global vision of the overall project goals. Furthermore, work on context aware applications in mobile computing and artificial intelligence [16] show that the user and his context have to be placed at the center of tool design in order to better answer his/her needs.

## 3.5 Multiple-Viewpoint Ontology-Enabled Traceability

To date models for multiple-viewpoint design have focused on devising mechanisms for explaining **how** the participating design concerns/viewpoints are connected. Some of these mechanism link domains at the model level. The most recent models link domains at the meta-model level. In both cases, however, support for understanding **why** they are connected does not exist.

We propose in this section to formulate a new model for ontology-enabled traceability that integrates the benefits of ontology-enabled traceability with meta-models for linking conceptual entities belonging to multiple design concerns. As illustrated in Figure 3.10, collections of design

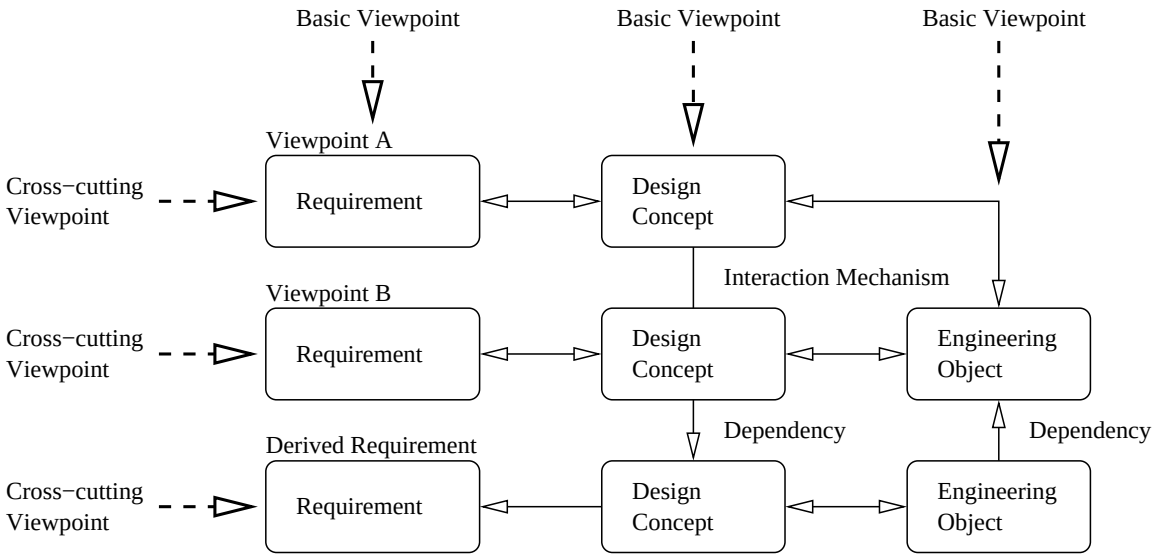



Figure 3.10: Extension of the proposed model to multiple-viewpoint design.

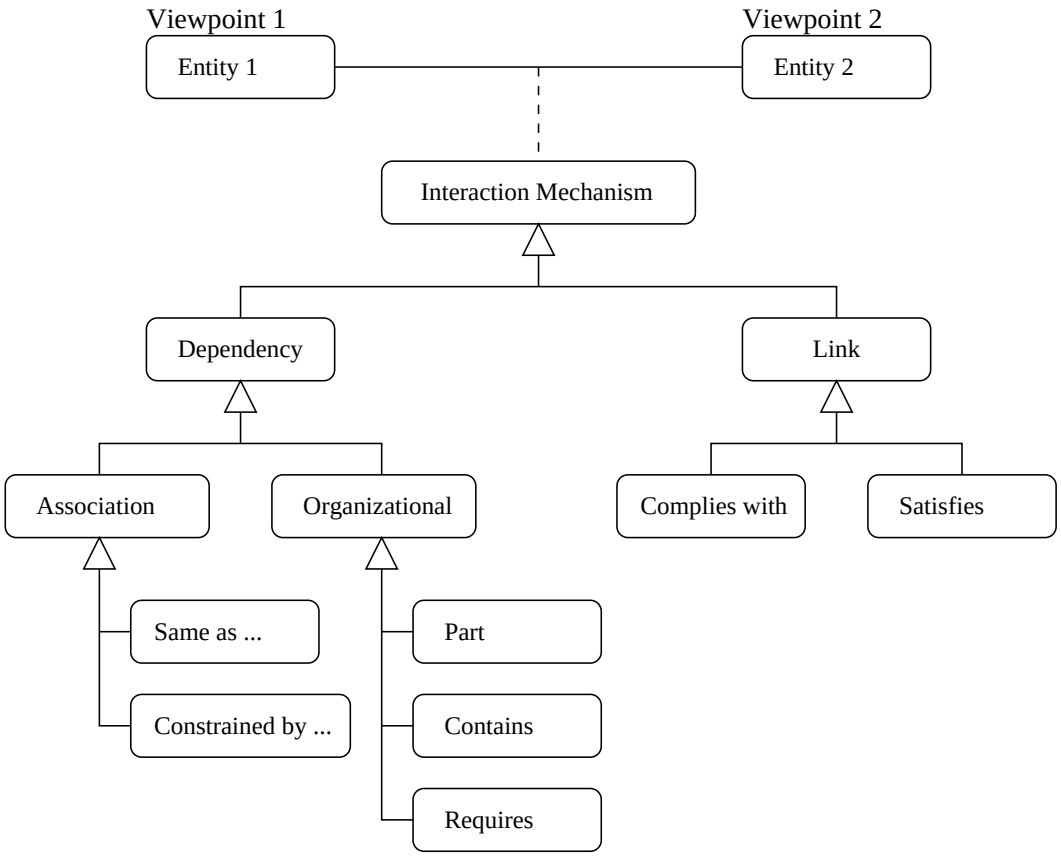



Figure 3.11: Class hierarchy for dependency relationships among design entities.

concerns associated with a domain will be modeled as graphs of design concept entities. Most of the graph edges will involve bi-directional relationships; however, dependency and association relationships will also be possible. Then as illustrated in Figure 3.11, individual design concepts will relate to other design concepts through a variety of interaction mechanisms. The association class `Interaction Mechanism` is a generalization of dependency and linkage mechanisms. Dependencies exist both within the ontology and, in the case of support for multiple viewpoints, to other ontologies.

Even though a group of ontologies may only provide a partial prescription for the functions and services that a system entity may need to provide it is still useful since a design can be checked with respect to the rules associated with the design concepts included in the model. Moreover, these domain-specific ontologies (meta-models) will be reusable across applications.

## 3.6 Simple Example: Renovation of a Wall in a House

To see how a multiple viewpoint implementation of ontology-enabled traceability might work out in practice, we now develop two models of viewpoint integration for a simple application: installation of a window into a load-bearing wall, the latter being part of a house. The two models will be kept as simple as possible by considering only architectural and structural engineering viewpoints, along with mechanisms for their interaction and potential conflict in the satisfaction of design concerns. The first model will be based on the work of Fences and co-workers in the 1990s (see Figures 3.8 and 3.9). The second model will build upon ideas illustrated in Figures 3.10 and 3.11.

**Wall Renovation Process and Modeling of Design Concerns.** Let us assume that a load-bearing wall in a house contains a door, but the neighboring space is too dark. An architect decides that the problem can be fixed by installing a window. This process is illustrated along the top of Figure 3.12.

From an architectural perspective, the wall helps to define a space which, in turn, will support a prescribed function for the occupants of that space (e.g., a room). A doorway provides access to the occupants and a window provides ambient light. The latter is also a means to providing fresh air to the occupants, thereby improving their comfort. Since the wall is load bearing, part

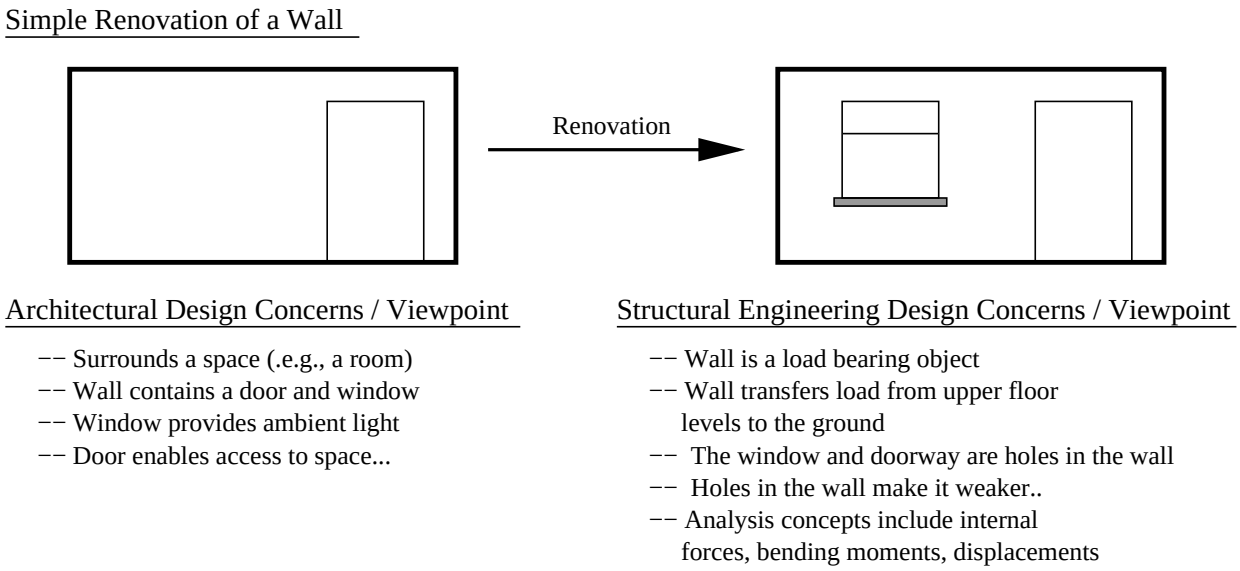



Figure 3.12: Schematic for installation of a window in a house wall and architectural and structural engineering design concerns.

| **Architectural Design Concerns** | **Representations** |
|---|---|
| • Functionality of the occupants<br>• Layout and arrangement of spaces.<br>• Elements used to define spaces<br>• Assignment of functionality to spaces<br>• Aesthetics<br>• Comfort | • Bubble diagrams<br>• Adjacency graphs<br>• Floorplans<br>• Elevation Views |
| **Structural Engineering Design Concerns** | **Representations** |
| • External loads, gravity loads, live loads.<br>• Selection of beams, columns, load-bearing walls.<br>• Assessment of an object's ability to transmit forces.<br>• Measurement of internal forces and displacements. | • Plan and elevation views.<br>• Finite element models. |

Table 3.1: Summary of viewpoints and representations for the building architecture and structural engineering disciplines.

of its purpose will be to provide a pathway for safe transmission of gravity forces to foundation. Structural engineers are responsible for making sure that the wall will have sufficient strength for this to occur, and to keep displacement and stability concerns within permissible limits. For this application, the architectural and structural engineering viewpoints are not only coupled through the size and positioning of the new window, but are in conflict. This tension arises because a large window may provide superior levels of ambient light, but the corresponding decrease in wall strength may be too great. A summary of these issues and corresponding representations and methods of analysis are summarized in Table 3.1.

**Model 1. Linking Architectural/Structural Engineering Design Concerns at the Model Level.**

Figure 3.13 is a schematic for capturing the dependency of architectural and structural engineering design concerns at the model level.

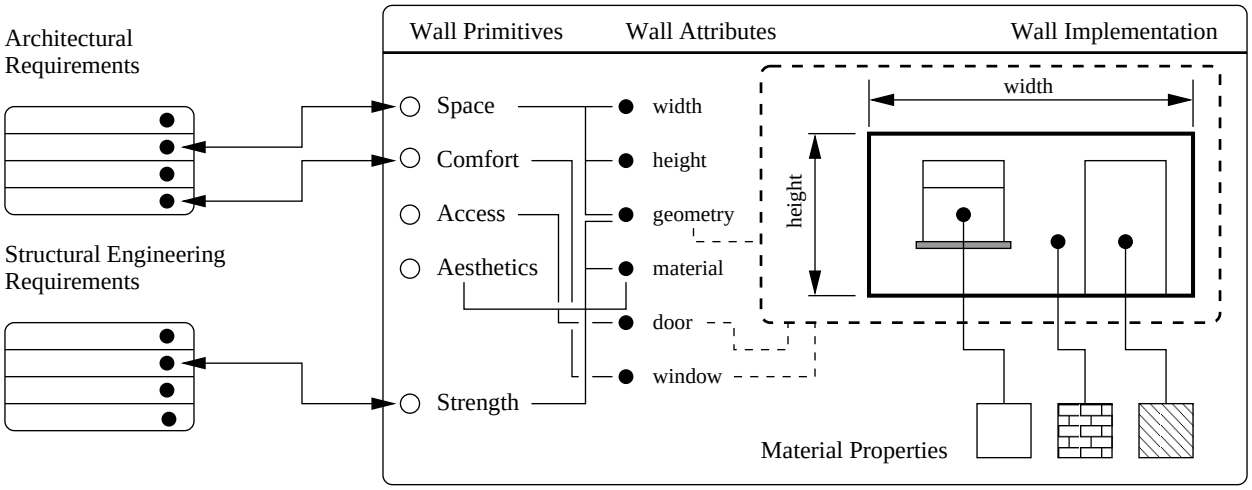



Figure 3.13: Multiple conceptual interpretations of a single design object.

As suggested by Fenves and co-workers, the wall's measures of effectiveness can be succinctly represented by Wall Primitives which, in turn, can be traced to wall attributes and aggregated groups of wall attributes. For example, one of the wall's primary architectural purposes is to participate in the definition of a space (e.g., a room). The characteristics of the space (e.g., its shape and size) will depend on the wall geometry; here we use the term "geometry" as a reference to a collection of lower-level geometric and topological quantities. We also assume that occupant comfort will be affected by the presence (or lack thereof) of a window. Then, in turn, the window

dimensions and positioning will affect the wall geometry. Access and Aesthetics primitives are tied to the existence of a door and choice of material. Finally, we assume that structural engineers are only interested in the wall strength which, in turn, depends on the wall geometry and choice of material.

**Assessment.** This simple example links the design concerns of architects and structural engineers at the object level, in this case, a singular wall. One key benefit of this approach is that models need not be complete in order for the approach to be applied. Our simplified model only shows a snapshot in the development. Although not explicitly captured by this model, groups of walls can be assembled into a wall system. And wall systems can be assembled into floorplans, wings of a building, and so forth. In fact, the mere existence of many organizational perspectives of a building is one of the main reasons buildings are interesting from a systems modeling perspective. The corresponding extension of Wall Primitives might be graphs of Building primitives. As far as we know, no tools exist for helping an engineer visualize and work with these graphs.

**Model 2. Linking Architectural/Structural Engineering Design Concerns through Ontologies.**

Our formulation for model 2 assumes that the most important design concepts and dependencies among concepts can be represented at the meta-model (or ontology) level.

Figure 3.14 shows simplified ontologies for the architectural and structural engineering domains, together with the linkage of domain concerns through interaction mechanisms. Within the architectural domain, for example, the ontology provides an explicit description for how a wall fits into the wall system which, in turn, complements definitions for a space and room. Individual walls are a composition of material properties and wall geometry, and they may contain portals – portal is a generalized term for opening, window or doorway. The functional purpose of doors and window can be connected to occupant needs (e.g., access and comfort) through the use of dependency relationships. From a structural engineering perspective, the wall system needs have sufficient strength which, in turn, depends on the selection of material properties and the wall geometry. In this simplified scenario, the architectural and structural engineering domains are linked through notions of material and geometry. In fact, in both viewpoints they are exactly the same thing.

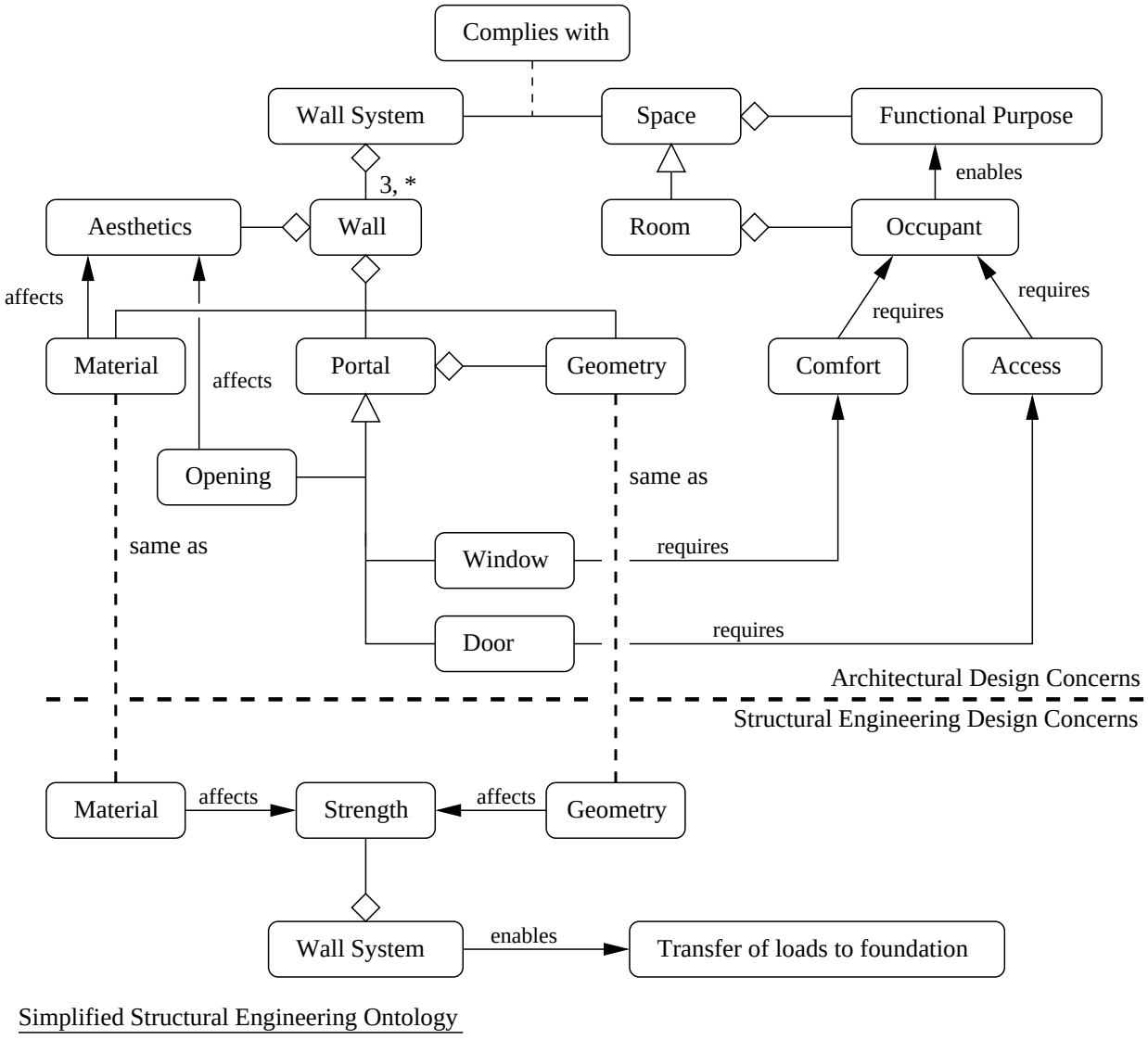



Figure 3.14: Linking of architectural and structural engineering ontologies. Linkages to libraries of design components (e.g., material selections) are not shown.

**Assessment.** There are several advantages in linking design concepts at the meta-model. First, model 2 is project neutral. Design concepts and relationships among design concepts can be reused across an entire family of project instances. A second key benefit is that it is much easier to show how a design concept entity relates to other entities. In other words, model 2 facilitates a "big picture" view of the essential concepts and relationships among concepts in a design situation.

# Chapter 4

# Software Architecture Design

For systems engineering design applications enhanced by ontology-enabled traceability, the software architecture design is concerned with the selection, modeling, and visualization of major software components and their connectivity into a networked architecture.

## 4.1 Network Architecture

As illustrated in Figure 4.1 below, we expect that software implementations will operate as a network of loosely coupled systems, connected only by traceability mechanisms and interfaces for communication of events and required data for tracking of dependencies and evaluation of design rules.

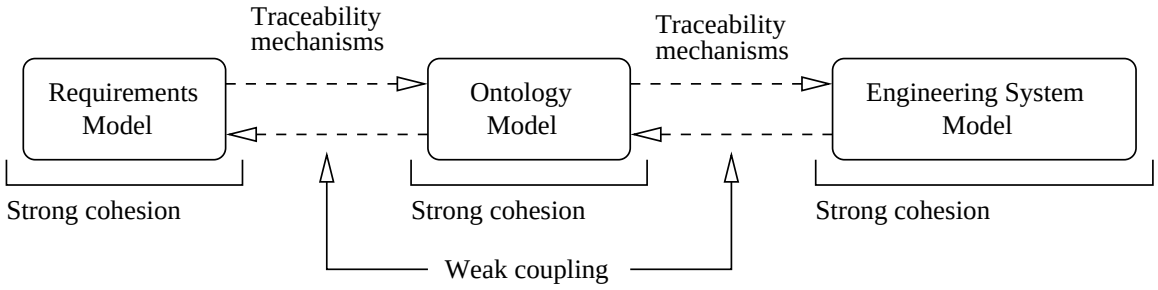



Figure 4.1: Overview of system architecture.

Looking forward 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. However, because traceability

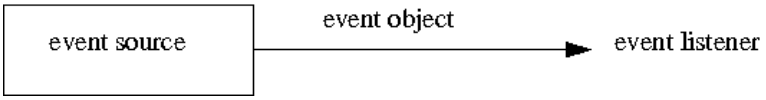



Figure 4.2: One Listener registered to One Source


Figure 4.3: Many Listeners registered to One Source

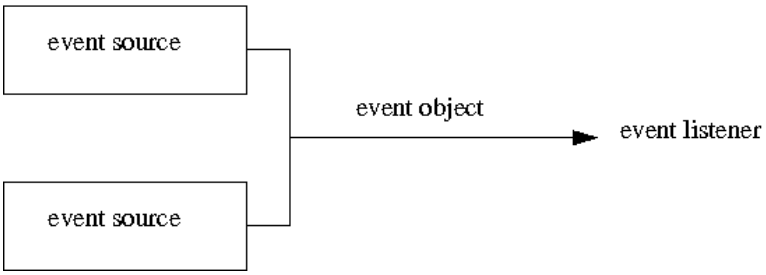



Figure 4.4: Many Listeners registered with Many Sources

relationships need to be bi-directional, connectivity also means: and (3) linking of objects in the engineering model back to UML classes (i.e., the ontology), and (4) linking UML classes (i.e., the ontology) back to the requirements.

Because requirements will change over time, a full implementation will need to support dynamic assembly and management of traceability relationships. Also, we anticipate that combinations of table and graph data structures will be used to store the data within each of the modules shown in Figure 4.1.

## 4.2 Delegation Event Model

Traceability connectivity and communication mechanisms 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. Pub-

lishers 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 [36].

## 4.3 Requirements-Ontology-Engineering Model Connectivity

The requirements to ontology to engineering model pathway is complicated by a chain of many-to-many relationships; see Figure 3.4. A single requirement may be satisfied through the implementation of one or more design concepts which, in turn, may correspond to multiple engineering entities. Looking backwards, a single design entity may help to satisfy multiple design concepts (e.g., it could be multi-functional) which, in turn, may trace back to multiple requirements. As a result, one can think of the overall model as three discipline-specific graphs, linked through a web of loosely-coupled dependencies.

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 4.5 and 4.6 show the step-by-step procedure for development, implementation and operation of ontology-enabled traceability in a design specification setting. Ontologies are described by concepts (classes) and relationships (e.g., association and inheritance relationships). For example, the simple ontology in Figure 4.5 states that an instance of A will contain instances of B and C. Multiplicity constraints could be added to constrain the number of instances of B and C with respect to A. Software implementations need 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 4.5), 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 4.6 shows the pathway of development for the processing of user events and design

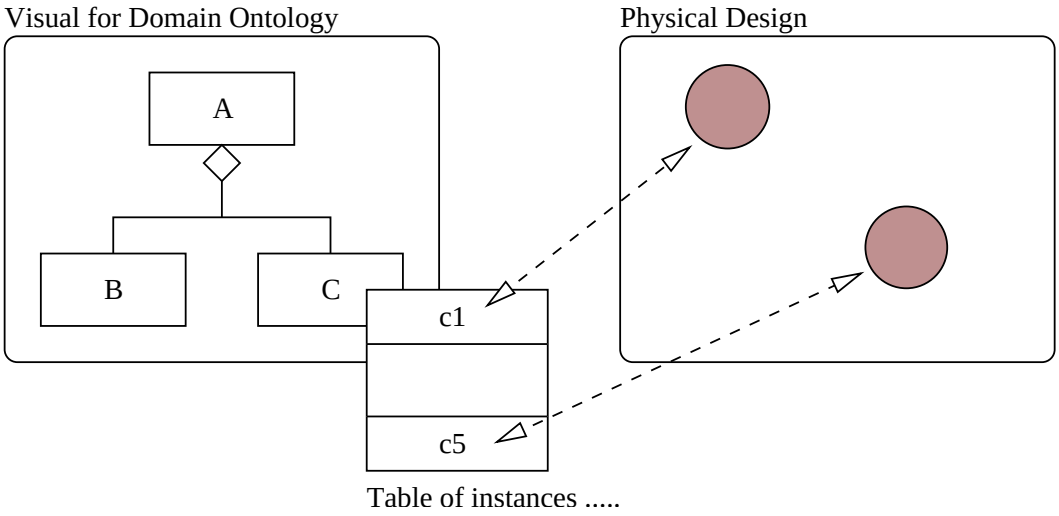



Figure 4.5: Connectivity between the Specification and Physical Models

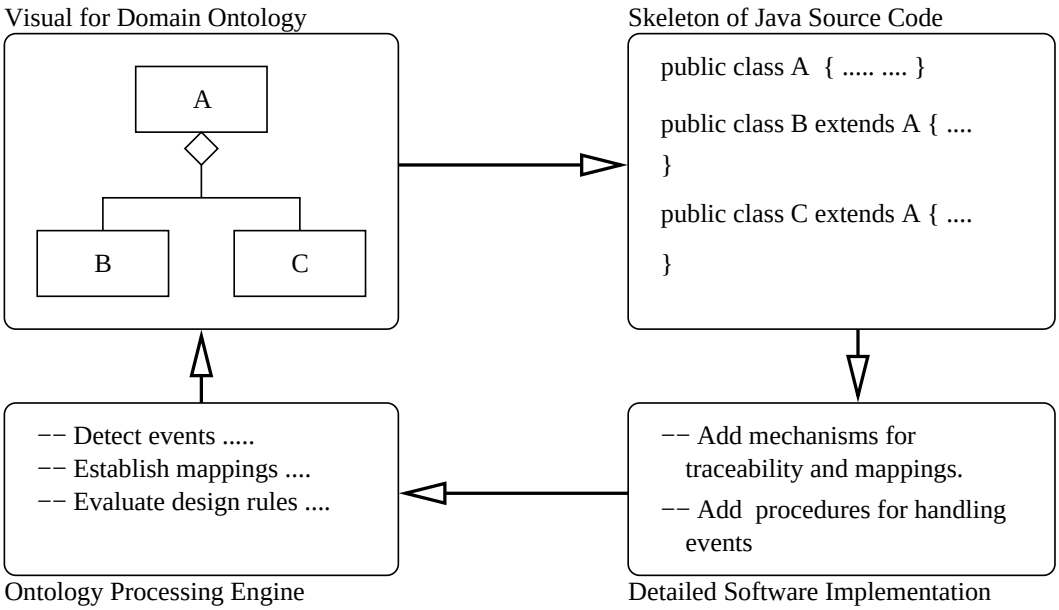



Figure 4.6: Step-by-Step Implementation of Ontology Processing Machine

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.

# Chapter 5

# Pilot Application: Design and Management of the Washington DC Metro System

This chapter describes our first prototype of ontology-enabled traceability, applied to the architecture-level design and management of the Washington DC Metro System. We have selected this application because it satisfies the dual objectives of being an important real-world engineering application, and yet, the underlying engineering model is not too complicated. In fact, as we will soon see, the metro systems architecture can be viewed simply as a geographic specialization of a graph structure.

The prototype application focuses exclusively on architecture-level design concerns of the as-built system from two viewpoints: (1) mathematical representation, and (2) transportation design. As such, physical and sociological-political ramifications of decisions on the metro station location and track routing are omitted, as are details of the track infrastructure at train stations (e.g., track sections; platforms), and train behavior, scheduling and control.

**The Washington DC Metro System**

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.


Figure 5.1: Map of the Washington DC Metro System

Figure 5.1 shows the map of the Metro System. The five Metro System lines cover the District of Columbia; the suburban Maryland counties of Montgomery and Prince George's; the Northern Virginia counties of Arlington, Fairfax and Loudoun; and the Virginia cities of Alexandria, Fairfax and Falls Church [66].

## 5.1 Framework for Metro System Design and Management

Modern railway systems are a complex intermingling of traditional infrastructure with electronics and telematics (i.e., GIS and GPS) [47]. To keep the complexity of design concerns in check, railway system design procedures strive to separate the underlying infrastructure (e.g., track profile and layout) from operational (e.g., schedule and capacity) and control (e.g., sequencing of switching and crossings) concerns. In systems engineering terminology the track infrastructure and railway vehicles define the systems structure. System behavior is defined by the operations and control. The first and most important priority is to ensure that all operations are completely safe. Then with safety concerns satisfied, schedules, capacity and switching operations are designed to maximize available capacity and minimize delays, subject to cost and performance constraints,

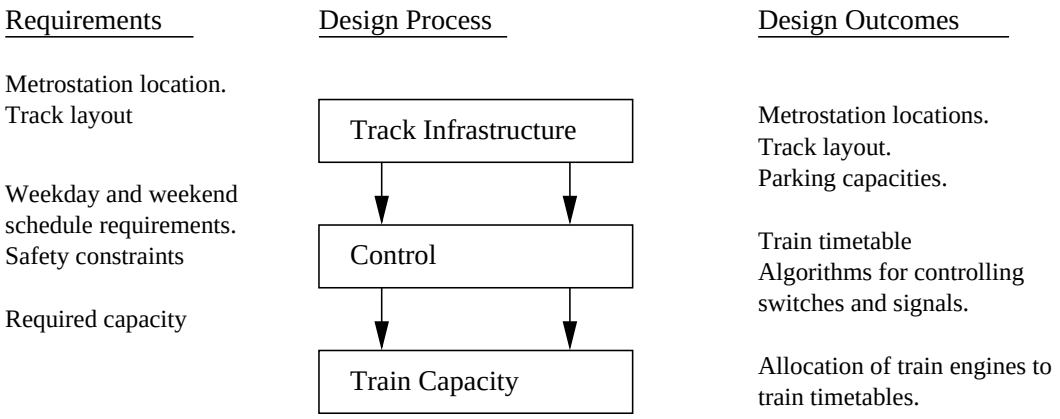



Figure 5.2: Flowdown of requirements and design outcomes in a top-down down development process.

Figure 5.2 shows the sequence of developments and flowdown of requirements in a (simplified) top-down development process. The development process begins with decisions on track infrastructure (e.g., positioning of metro stations; track layout). Then issues of scheduling (e.g., weekday and weekend departure and arrival times for trains) and train control (e.g., routing trains through railway stations) are handled. The primary purpose of a railway control system is to prevent events from happening that could lead to an unsafe system state. Generation of a "train timetable" is often complicated by highly utilized and intertwined railway networks with many connections between trains [68]. Since many section of the track will operate as a shared resource (meaning that different trains will use the same section of track at different times), strategies for scheduling and control must guarantee that all safety constraints are met. The most straightforward approach to achieving this objective is to implement centralized control algorithms that: (1) have access to the global state of the system, and (2) verify correctness of system operations through formal analysis. Finally, decisions are made on train selection to satisfy requirements on scheduling and passenger capacity. A complete study would also generate a mix of best-engine allocations for a number of fleet alternatives [20].

## 5.2 Graphical User Interface Design

Figure 5.3 shows the layout of windows in the prototype software implmentation 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

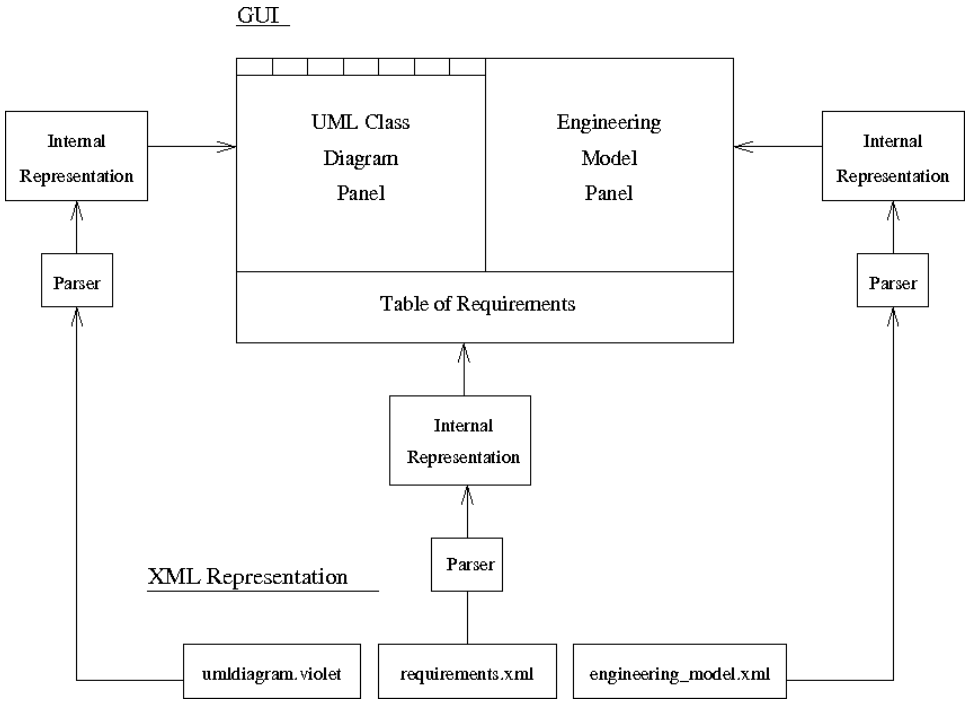



Figure 5.3: Graphical user interface design and connection to XML persistent storage.

table of requirements, a UML diagram panel for the application ontology, and an engineering model panel containing the model of the system. As such, the interface provides cross-cutting viewpoint of development. A stakeholder should be able to any aspect of the graphical interface and see how features at one stage of development relate (i.e., via dependencies, etc) to features at another stage of development.

As we will soon see, 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 entities.

## 5.3 Requirements-Ontology-Engineering Software Prototype

Figure 5.4 is a screendump of the Washington DC Metro System Requirements-Ontology-Engineering graphical user interface.

The software prototype has a user interface and XML input/output consistent with the

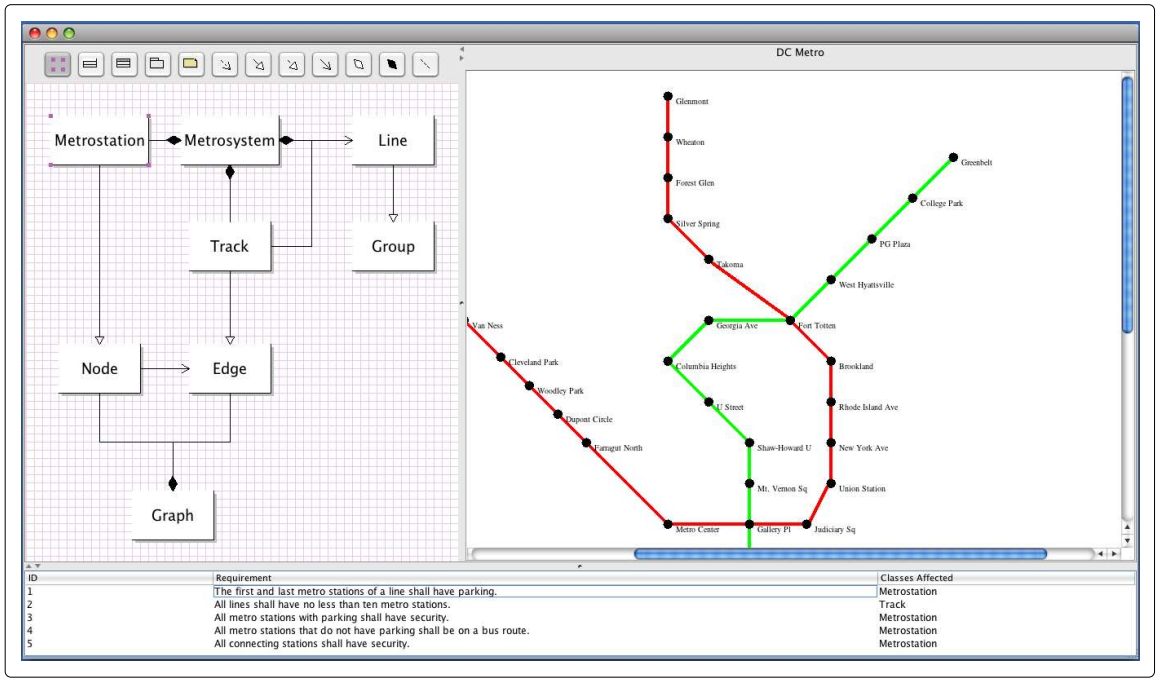



Figure 5.4: Requirements-Ontology-Engineering GUI for the Washington DC Metro System

spcifications of Figure 5.3, and inspired in part by the open source graph editor framework for Violet, a UML editor developed by Cay Horstmann [28]. Violet supports the drawing of class diagrams, sequence diagrams, use case diagrams, state diagrams and object diagrams – the underlying implementation relies on a framework for creating and editing graph data structures. In the requirements-ontology-engineering software connectivity relationships are also 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.

**System Requirements.** The metro system design is modeled with only five requirements:

**Req. 1.** The first and last mtro stations of a line shall have parking,

**Req. 2.** All lines shall have no less than ten mtro stations.

**Req. 3.** All mtro stations with parking shall have security,

**Req. 4.** All mtro stations that do not have parking shall be on a bus route, and

**Req. 5.** All connecting stations shall have security.

The corresponding XML representation is illustrated in Figure 5.5.

```
<requirements>
    <requirement id="1">
        <text>The first and last metro stations of a line shall have parking.</text>
        <class>Metrostation</class>
    </requirement>
    <requirement id="2">
        <text>All lines shall have no less than ten metro stations.</text>
        <class>Track</class>
    </requirement>
    <requirement id="3">
        <text>All metro stations with parking shall have security.</text>
        <class>Metrostation</class>
    </requirement>
    <requirement id="4">
        <text>All metro stations that do not have parking shall be on a bus route.</text>
        <class>Metrostation</class>
    </requirement>
    <requirement id="5">
        <text>All connecting stations shall have security.</text>
        <class>Metrostation</class>
    </requirement>
</requirements>
```

Figure 5.5: XML Representation for the Requirements

The XML datafile for the requirements is parsed and inserted into the table (i.e., java JTable) shown along the bottom of Figure 5.4.

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.

Notice that these requirements only cover design concerns related to the system structure and the existence of attributes whose values can be used in requirements evaluation. A more comprehensive study – see comments in future work – would also include requirements associated with train behaviors and time-driven schedules.

**Metro System Ontology.** The top left-hand panel shows the mtro system ontology represented in a UML class diagram format. The ontology diagram serves two stakeholder perspectives.

From a mathematical standpoint, the transportation network is simply a graph of nodes and edges attached to a spatial surface. A node can be characterized by its name and geographical

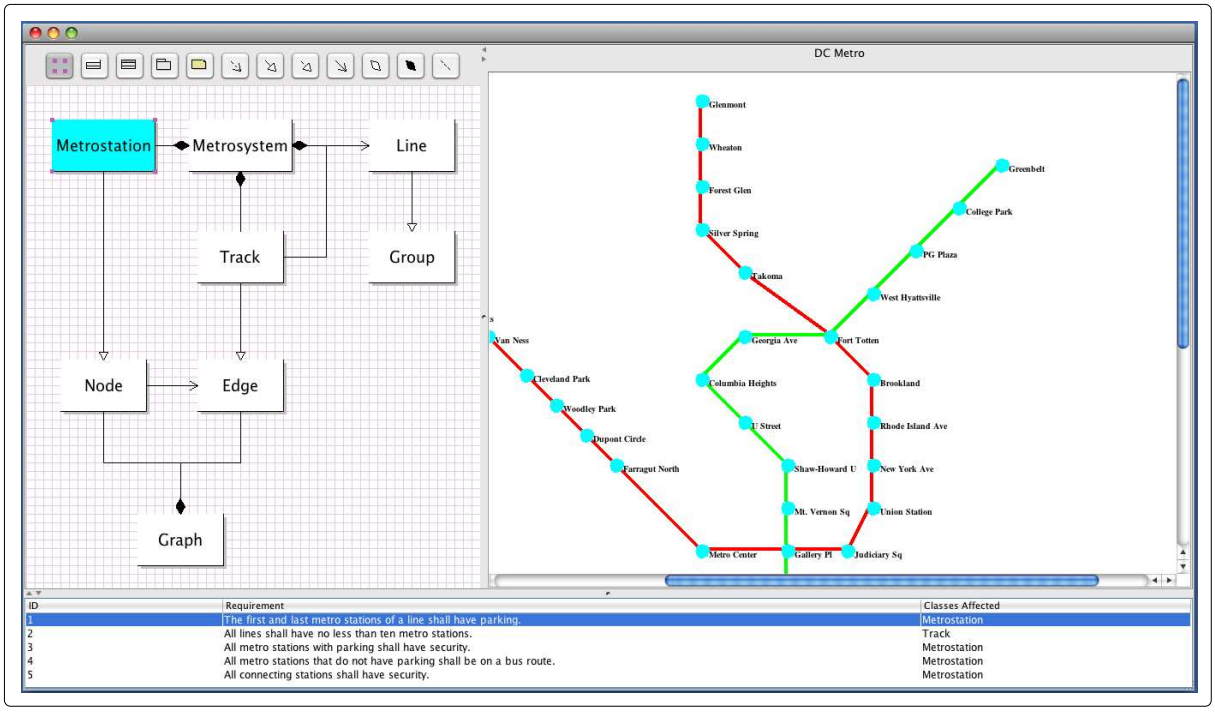



Figure 5.6: Tracing a requirement to the UML class diagram and onto the engineering model.

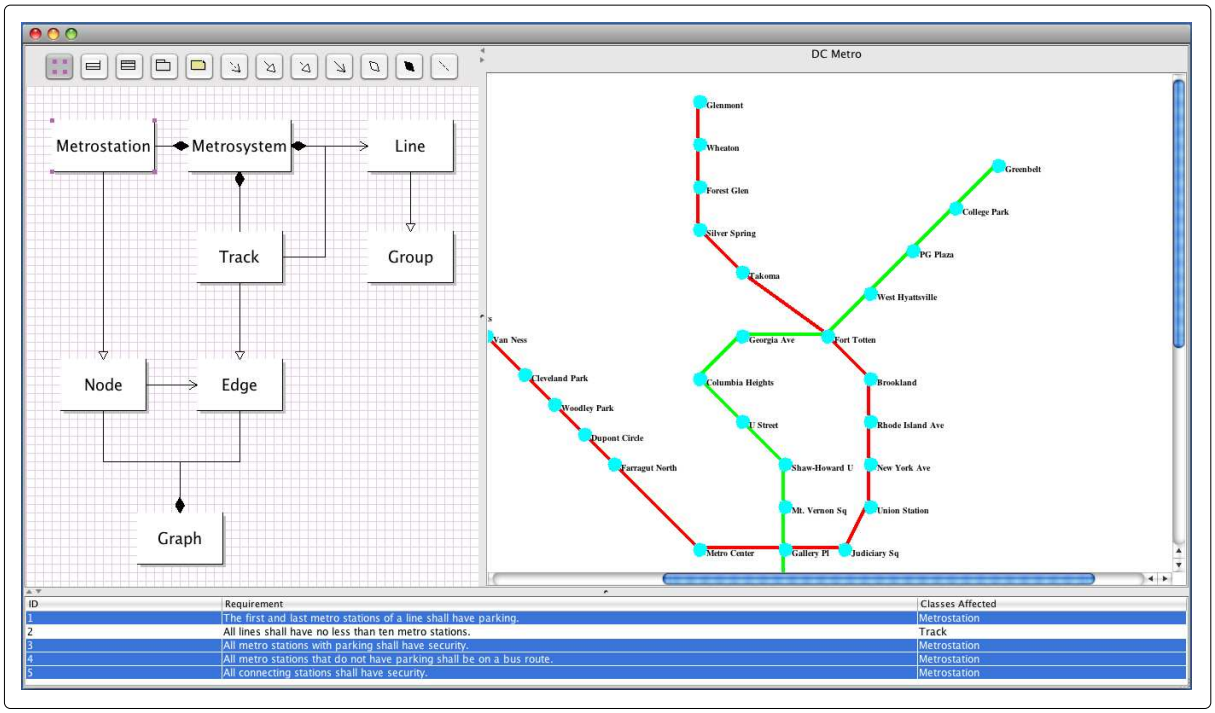



Figure 5.7: Graphical display of requirements and engineering model objects associated with the MetroStation class.

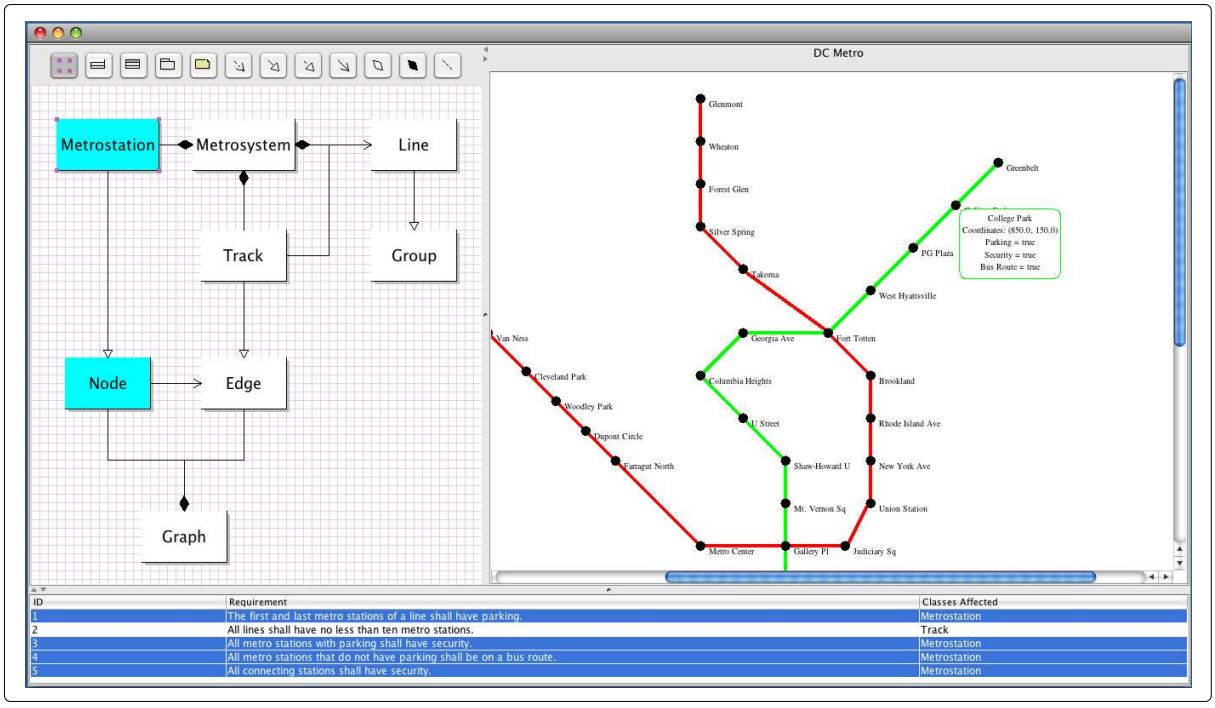



Figure 5.8: Graphical display of requirements and ontology classes associated with the College Park Metro Station.

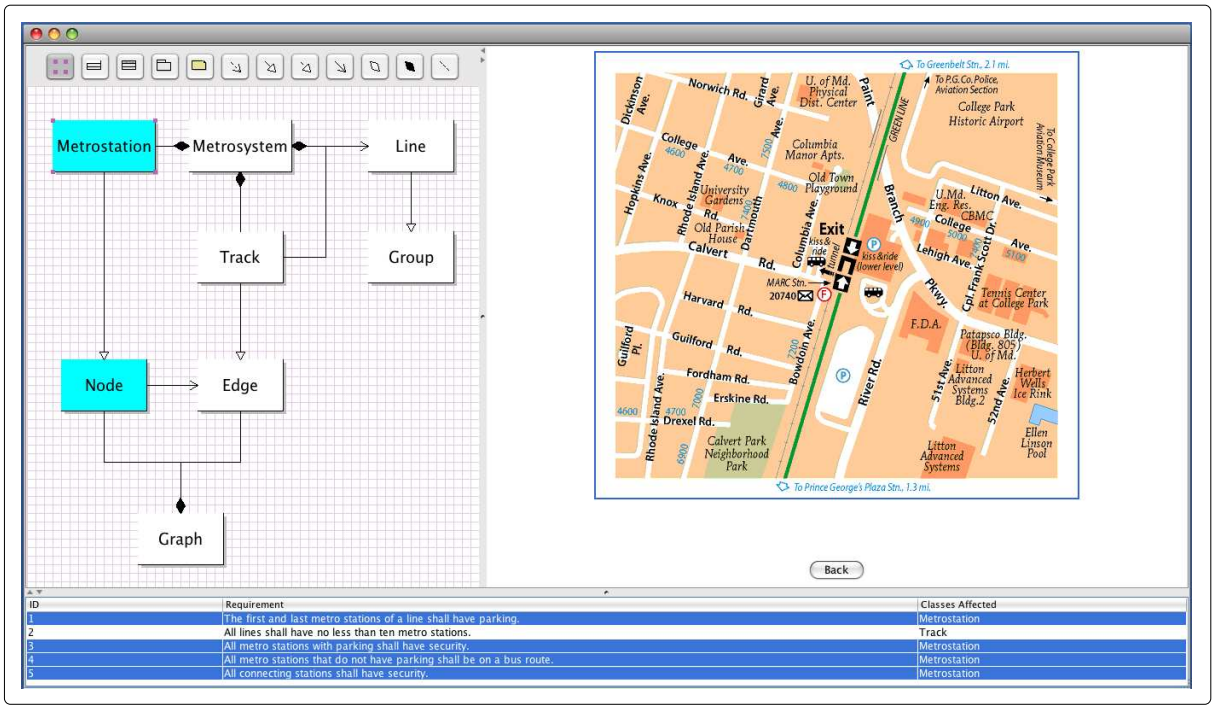



Figure 5.9: Graphical display of requirements and ontology classes traced to a detailed map of the College Park Metro Station.

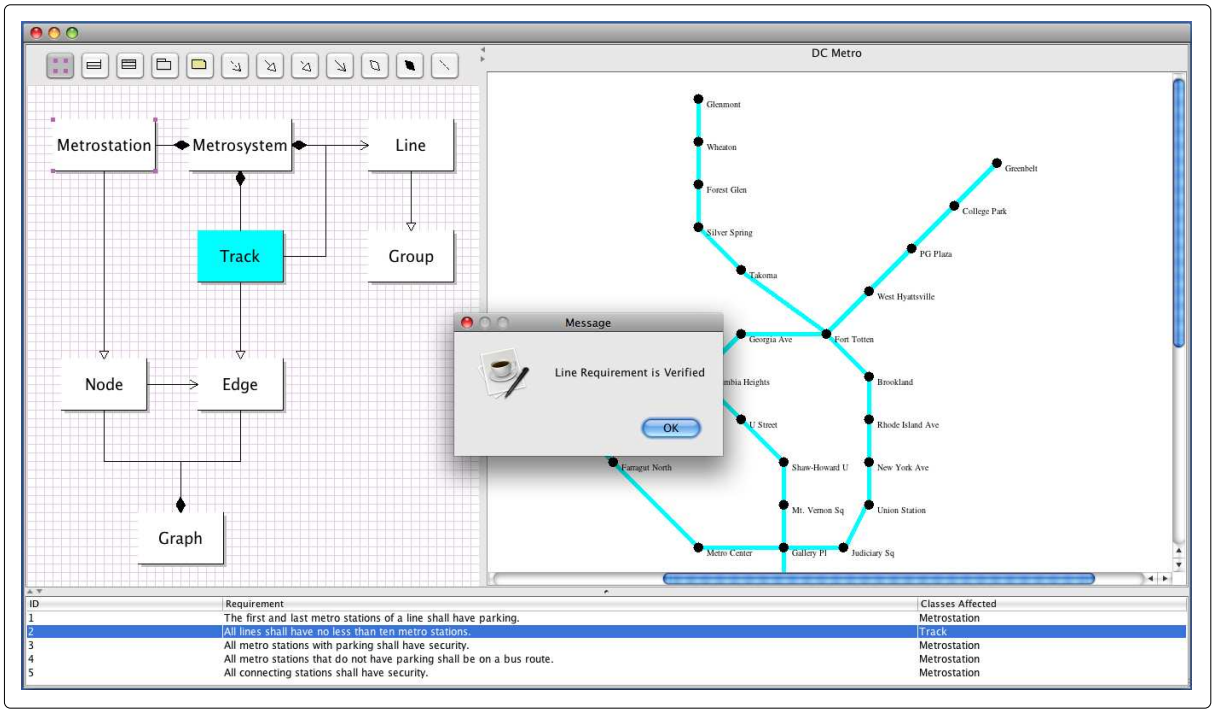



Figure 5.10: Implementation of rule checking for the track requirement.

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. Metro 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 – passengers talk about catching a green line train to the College Park Metro Station, but in reality neither the trains nor the track are actually painted green.

## 5.4 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.

## 5.5 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 vent from the requirement. Double clicking a requirement triggers the verification of that requirement against the engineering model. For example, the first requirement (end of line mtro 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 procss will be highlighted.

## 5.6 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.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

The key contributions of this work are twofold: (1) A new methodology and system for satisfying requirements with ontology-enabled traceability mechanisms, and (2) An architectural framework for linking discipline-specific dependencies through interaction relationships at the meta-model (or ontology) level. In state-of-the-art traceability mechanisms, requirements are connected directly to design objects. Here, in contrast, we ask the question: What design concept (or family of design concepts) should be applied to satisfy this requirement? Solutions to this question establish links between requirements and design concepts. Then, it is the implementation of these concepts that leads to the design itself.

The proposed methodology offers several benefits not possible with state-of-the-art procedures. First, procedures for design rule checking may be embedded into design concept nodes, thereby creating a pathway for system validation and verification processes that can be executed early in the systems lifecycle where errors are cheapest and easiest to fix. Second, the proposed model provides a much better big-picture view of relevant design concepts (see the examples in Chapters 3 and 5) and how they fit together, than is possible with linking of domains at the model level. And finally, the proposed procedures are automatically reusable across families of projects where the ontologies are applicable.

Our focus in this study has been on the development and preliminary evaluation of ontology-enabled traceability mechanisms for engineering design. As already noted, in design the predomi-

nant pathway of development is identification of design concepts that can help to satisfy requirements in a real-world implementation. It is important to note, however, that the proposed methodology can also support operations associated with systems management and health monitoring. One can imagine, for example, a network of sensors feeding streams of data to objects/entities in an engineering model (an abstraction of a real-world implementation). Then, in turn, these objects/entities will be connected to design concepts which can support the execution of rule checking processes to find anomalies and unacceptable deviations in behavior. In a working implementation, system operators would be notified that something has gone awry through the graphical visualization of requirements violations.

## 6.2 Future Work

As a general observation, the implementation of methods and tools for ontology-enabled traceability is complicated by a chain of many-to-many relationships connecting requirements, design concepts and engineering objects. Each design concern can be modeled as a graph; the complete system model is a collection of weakly coupled graphs. To keep the details of implementation for our prototype application as simple as possible, all of the modeling dependencies were hard coded. An improved implementation would provide support for the efficient and scalable management of these links. Because we were primarily interested in making a point, our prototype application also employed a simplified representation for ontologies covering multiple design perspectives. There is a need to explore ways of treating domain ontologies as individual entities, yet, show the dependencies among ontologies that are important for design. We surmise that a three-dimensional visualization of ontologies and their connections might be useful. There is also a question of trust that needs to be considered. In order for the proposed methodology and system to actually reduce the likelihood of system failures, system-level designs need to faithfully represent both the stakeholder needs and the capabilities of the participating application domain(s). Moreover, the ontology models need to be accurate, complete, conflict free and minimal (i.e., no redundancy) [54].

Our ontology-enabled traceability model is now being extended along the lines of the annotations accompanying the screendump in Figure 6.1. 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, and even the value of attributes within states of behavior models. This capability will provide a direct

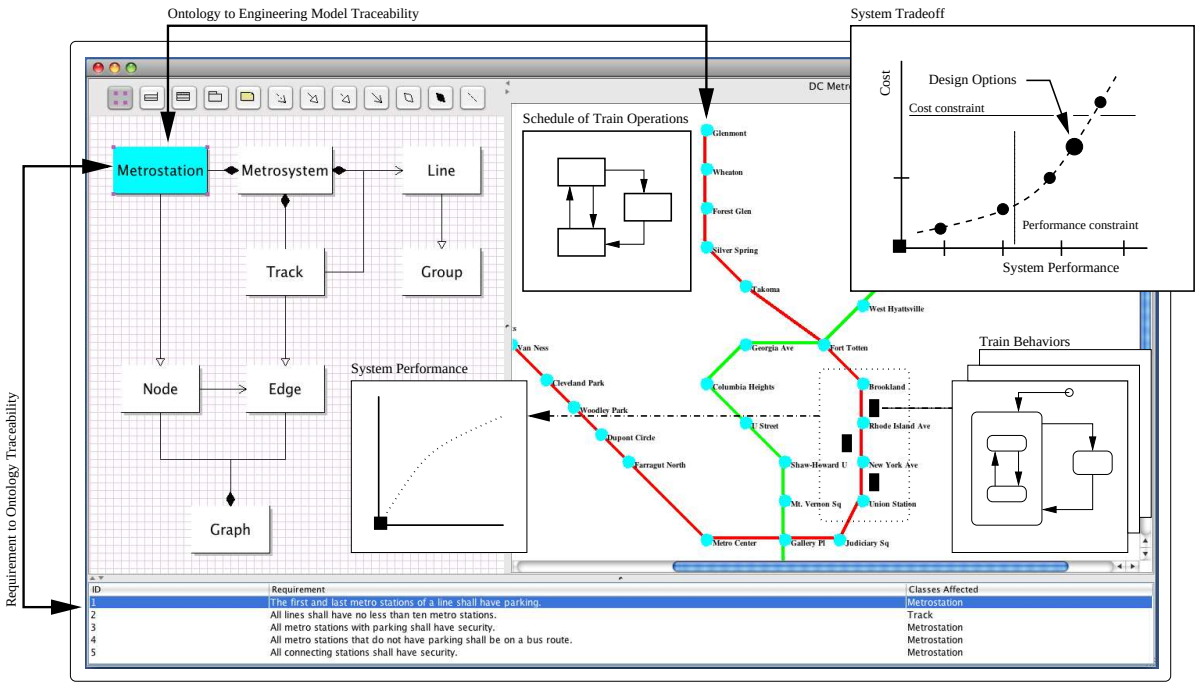



Figure 6.1: Annotated prototype for requirements-ontology-engineering traceability in the Washington DC Metro System. Here, an end-of-line parking requirement is associated with the MetroStation ontology which, in turn, traces to all of the metro station instances in the Washington DC Metro System.

pathway from requirements to evaluation of performance attributes which, in turn, will allow for tradeoff studies.

Looking forward, the small table of requirements will be replaced by PaladinRM, an interactive java-based tool for working with large graphs of engineering requirements [2]. We will also investigate the feasibility of replacing UML diagrams with the Web Ontology Language (OWL) and reasoning procdures driven by the Semantic Web Rule Language (SWRL). Fundamental research is needed to understand the extent to which: (1) various kinds of relationships can be formally represented, and (2) traceability pathways can be automatically assembled among design entities and viewpoints. This will require precise definition of entities in the ontology meta-meta-model (see Figure 3.3), so that chains of reasoning and rule checking implied by relationships among conceptual domains will work correctly.

# Bibliography


[1] Austin M.A. Information-Centric Systems Engineering. *Lecture Notes for ENSE 621-622-623*, 2005. Institute for Systems Research, University of Maryland, College Park, MD.

[2] Austin M.A., Mayank V., and Shmunis N. PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design. *Systems Engineering: The Journal of the International Council on Systems Engineering*, 9(2):129–145, May 2006.

[3] Baclawski K., Kokar M.K., Kogut P., Hart L., Smith J., Holmes W., Letkowski J., and Aronson M.L. Extending UML to Support Ontology Engineering for the Semantic Web. *UML2001*, 2001.

[4] Baier C. and Katoen J.P. *Principles of Model Checking*. MIT Press, Cambridge, MA 02142, 2008.

[5] Balasubramaniam R., Jarke M.,. Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(1), January 2001.

[6] Bentley K., 2003. Does the Building Industry Really Need to Start Over? – A Response from Bentley to AutoDesk's BIM/Revit Proposal for the Future.

[7] Berners-Lee T., Hendler J., Lassa O. The Semantic Web. *Scientific American*, pages 35–43, May 2001.

[8] Broderbund. 3D Home Architect Design Suite. *Deluxe 6*, 2004. See http://www.broderbund.com.

[9] Calvano C.N., and John P. Systems Engineering in an Age of Complexity. *Systems Engineering*, 7(1):25–34, 2004.

[10] Cao X., He Z., and Pan Y. Automated Design of House-Floor Layout with Distributed Planning. *Computer-Aided Design*, pages 213–222, 1990.

[11] Chun H.W., and Lai M.K. Intelligent Critic System for Architectural Design. *IEEE Transactions on Knowledge and Data Engineering*, 9(4), July/August 1997.

[12] CORE. See http://www.vitechcorp.com/productline.html. 2009.

[13] Cranefield S. Networked Knowledge Representation and Exchange using UML and RDF. *Journal of Digital Information*, 1(8), February 2001.

[14] Cranefield S. UML and the Semantic Web. In *In Proceedings of the International Semantic Web Working Symposium*, Palo Alto, 2001. See http://www.semanticweb.org/SWWS/program/full/paper1.pdf.

[15] de Vries B., Jessurun A.J. and van Wijk J.J. Interactive 3D Modeling in the Inception Phase of Architectural Design. *The Eurographics Association*, 2001.

[16] Doekhorn C.P. Towards a Services Platform for Context-Aware Platforms, Master of Science Degree in Telematics. Technical report, University of Twente, Enschede, The Netherlands, 2003.

[17] Dynamic Object Oriented Requirements System (DOORS). See http://www.telelogic.com/products/doorsers/doors/. 2009.

[18] Downs L. Interchange Format for Symbolic Building Design. Dissertation submitted in partial satisfaction for the MS degree in Computer Science, University of California, Berkeley, CA, 1999.

[19] Eeles P. and Cripps P. *The Process of Software Architecting*. Addison-Wesley, 2010.

[20] Florian M., Bushell G., Ferland J., Guerin G., and Nastansky L. The Engine Scheduling Problem in Railway Network. *INFOR*, 14(2), June 1976.

[21] Fogarty K. System Modeling and Traceability Applications of the Higraph Formalism. *MS Thesis in System Engineering, Institute for Systems Research*, May 2006.

[22] Fogarty K. and Austin M.A. System Modeling and Traceability Applications of the Higraph Formalism. *Systems Engineering: The Journal of the International Council on Systems Engineering*, 12(2):117–140, Summer 2009.

[23] Geroimenko V., and Chen C. (Eds). *Visualizing the Semantic Web: XML-based Internet and Information Visualization*. Springer, 2003.

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

[25] Gotel O.C.Z., Marchese F.T., and Morris S.J. On Requirements Visualization. In *In Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007)*, 2007.

[26] Halin G., Kubicki S., Bignon J.C. *A Multi-View Cooperative Platform for Building Construction*. MAP CRAI, Architecture School of Nancy, Nancy Cedex, France, 2006.

[27] Hendler J. Agents and the Semantic Web. *IEEE Intelligent Systems*, pages 30–37, March/April 2001.

[28] Horstmann C. *Object-Oriented Design and Patterns, Second Edition*. John Wiley and Sons, New York, 2006. (See pages 334-352.).

[29] IBM White Paper. Dynamic Infrastructure Helping to Build a Smarter Planet: Delivering Superior Business and IT Services with Agility and Speed. *IBM Thought Leadership*, February 2009.

[30] 2009. IBM Telelogic SLATE: See http://www.craiglarman.com.

[31] IEEE 1471, Recommended Practice for Architectural Description of Software Intensive Systems, IEEE Std 1471-2000. For details, see http://standards.ieee.org/reading/ieee/ std_public/ description/se/1471-2000_desc.html (Accessed April 17, 2010), 2000.

[32] Jackson D. Dependable Software by Design. *Scientific American*, 294(6), June 2006.

[33] Jones, N., 2004. Flawed Drawings caused Spacecraft Crash: Upside-Down Switches stopped Parachutes from Opening, news@nature.com.

[34] Kharrufa S., Saffo A., and Mahmood W. Developing CAD Techniques for Preliminary Architectural Design. *Computer-Aided Design*, pages 213–222, 1985.

[35] Kogut P., Cranefield S., Hart I., et al. UML for Ontology Development. *The Knowledge Engineering Review*, 17(1).

[36] Larman C. Implementing the Java Delegation Event Model and JavaBeans Events in C++, April 1999. See http://www.craiglarman.com.

[37] Liang V.C., and Paredis C.J.J. A Port Ontology for Conceptual Design of Systems. *Transaction of the ASME*, 4, September 2004.

[38] Madeen D.A., Palma R.M. Architectural AutoCAD: Drafting, Design and Presentation. 2001.

[39] Magee J.L., and Kramer J. *Concurrency: State Models and Java Programs (2nd Edition).* John Wiley and Sons, New York, 2006.

[40] Maier, M.W. Reconciling System and Software Architectures. *The Aerospace Coorporation*, 1998.

[41] 2002. MathML. Referenced on April 6, 2002. See hhttp://www.w3.org/Math.

[42] MATLAB. See http://www.mathworks.com (Accessed October 1, 2010).

[43] Modelica. See http://www.modelica.org (Accessed October 1, 2010).

[44] Muller D. Requirements Engineering Knowledge Management based on STEP AP233. 2003.

[45] 2000. OMG: Meta Object Facility (MOF) Specifications, 2000, Object Management Group. See http://www.omg.org/ and references therein.

[46] OpenDesign. TurboCAD Deluxe. *Version 10*, 2004. The deluxe edition handles 2d- and 3d drawings, definition of simple solids, and boolean operations. A professional edition is supported by the ACIS solid modeling package.

[47] Profillidis V.A. *Railway Engineering: Second Edition.* Ashgate, 2000.

[48] Rivard J., Fenves S.J., and Gomez N. An Information Model for Multiple Views of Buildings. In *Modeling of Buildings through their Lifecycle: Proceedings of CIB W78/TG10 Workshop, CIB Publication No. 180*, pages 248–259, Rotterdam, The Netherlands, 1995.

[49] Rosenman M.A., Gero J.S. Modeling Multiple Views of Design Objects in a Collaborative CAD Environment. *Computer-Aided Design*, 28(3):193–205, 1996.

[50] Sangiovanni-Vincentelli A. Automotive Electronics: Trends and Challenges. In *Presented at Convergence 2000*, Detroit, MI, October 2000.

[51] Sangiovanni-Vincentelli A., McGeer P.C., Saldanha A. Verification of Electronic Systems : A Tutorial. In *Proceedings of the 33rd Desgin Automation Conference*, Las Vegas, 1996.

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

[53] Sequin C. and Downs L. Symbolic CAD Tools for Architecture. Technical report, University of California, Berkeley, CA, 1997.

[54] Shanks G., Tansley E., Weber R. Using Ontology to Validate Conceptual Models. *Communications of the ACM*, 46(10):85–89, 2003.

[55] Sidorova N. Lecture Notes in Process Modeling. 2007. Department of Mathematics and Computer Science, Eindhoven University, Netherlands.

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

[57] 2002. Scalar Vector Graphics (SVG). Referenced on April 5, 2002. See http://www.w3.org/Graphics/SVG/Overview.html.

[58] SysML Partners, Systems Modeling Language. Note. This presentation describes the extensions to UML 2 for Systems Engineering. New diagrams types include "requirements" and "parametric" diagrams. November 2003.

[59] Tidwell D. *XSLT*. O'Reilly and Associates, Sebastopol, California, 2001.

[60] Tien J.M. Toward a Decision Informatics Paradigm: A Real-Time Information-Based Approach, to Decision Making. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 33(1), February 2003.

[61] Uchitel S., Kramer J., and Magee J. Incremental Elaboration of Scenario-Based Specifications and Behavior using Implied Scenarios. *ACM Transactions on Software Engineering and Methodology*, 13(1):37–85, January 2004.

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

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

[64] Whitney D.E. Why Mechanical Design cannot be like VLSI Design. *Research in Engineering Design*, 8:125–138, 1996.

[65] Wieringa R. A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. *ACM Computing Surveys*, 30(4), December 1998.

[66] 2006. WMATA Facts. See http://www.wmata.com/about/metrofacts.pdf. Accessed, December 2006.

[67] XML Stylesheet Transformation Language (XSLT). See http://www.w3.org/Style/XSL. 2002.

[68] Zwaneveld P.J., Kroon L.G., Stan P.M., and van Hoesel S.P.M. Theory and Methodology: Routing Trains through a Railway Station based on a Node Packing Model. *European Journal of Operations Research*, 128:14–33, 2001.

# Appendix A

# Appendices

## A.1 Architectural Design for Modern Building Environments

The key ideas in our proposed approach were initially flushed out by thinking about how they might apply to the domain of architectural and services design for modern building environments. As a focus for study this problem domain is appealing because it is easy to understand, yet, good solutions demand a team-based approach to development with input and coordination of activities from multiple disciplines.

**Modern Building Environments.** Modern building environments are highly multidisciplinary systems, serving many stakeholders over extended periods of time. Design solutions are assembled from concepts pertinent to well-known design concerns (e.g., behavior, structure, testing), contexts, and viewpoints [49]. Architectural design processes typically begin with the identification of enabled functionality and building services, followed by an assessment of required spaces and their organization/clustering. Constraints on performance and cost place bounds on these spaces. While the results of architectural design are most often represented as documents and blueprints, this is changing. The CTO of Bentley Systems, a leading provider of architectural design software notes that architectural/engineering firms need to move from "drawings" to "building information models," the latter being capable of representing and reasoning with graphical and non-graphical entities. Building information models are compelling because they enable processes for designing-in-context across disciplines and automatically enforcing standards. The resulting product is more correct by design [6].

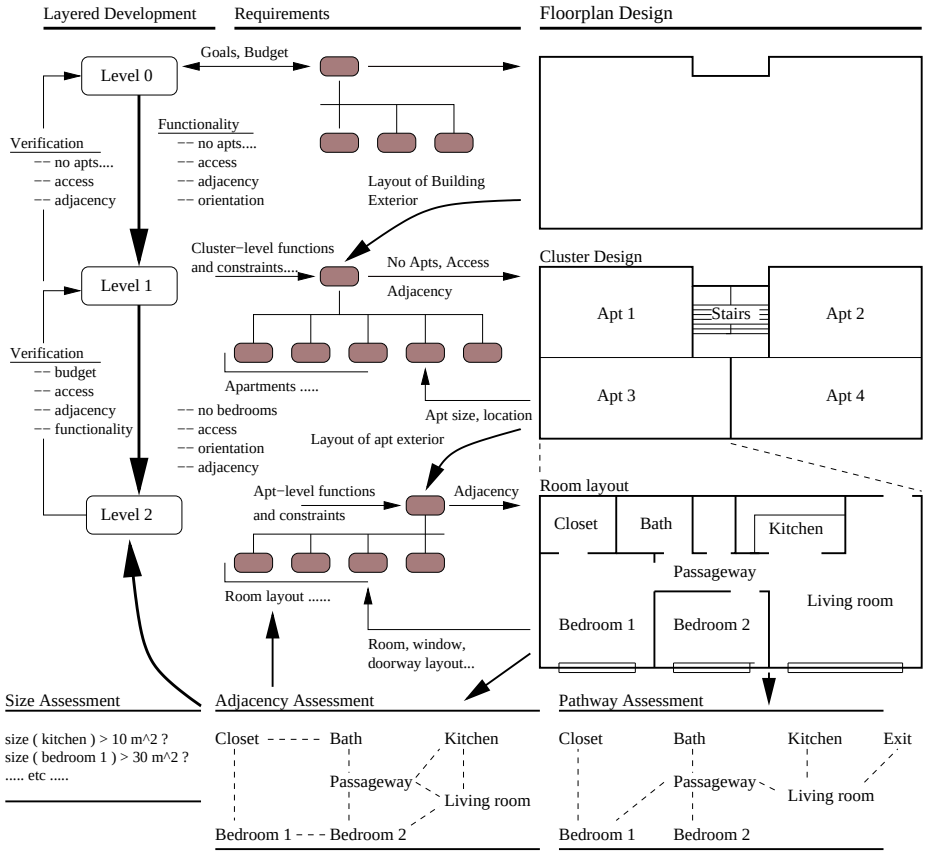



Figure A.1: Progressive Decomposition of Architectural Floorplans

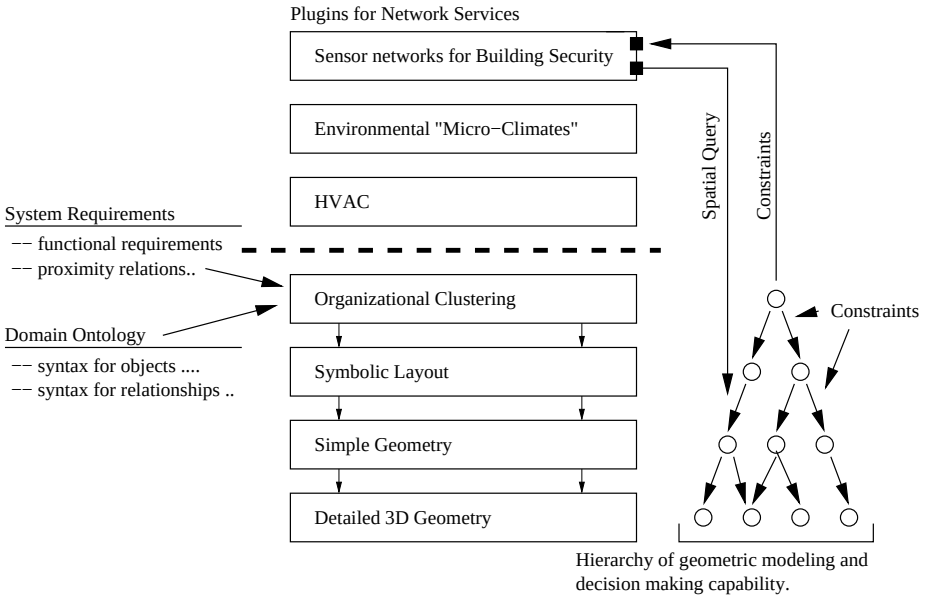



Figure A.2: Framework for Multi-Level Development for Building Architectures (Spatial Arrangements) Augmented with Network Services (Adapted from Downs and Sequin [18, 53])

From a systems modeling perspective, building behaviors are deceivingly simple; at the architectural level there are no obvious components where inputs are transformed into outputs, and traditional approaches to assembly of system behavior through composition of component functionalities does not seem to apply. In reality, while many of the components in a building may just sit there and do nothing, they are in fact necessary enablers of complex behaviors within building environments. Many building environment behaviors are, in fact, so complex that we do not even attempt to create detailed models of the enabled processes. Instead we step away from detailed models and simply provide spaces to support functionalities and leave the details of implementation up to the building occupants.

**Established Approach to Architecture Development.** Figures A.1 and A.2, show the established pathway of architectural floorplan development. During the earliest stages of design, architectural concerns are directed toward development of functional requirements and identification of relevant design rules, and economic and legal restrictions. The progressive transformation from required functional to constructive entities is very much a creative process. Initially, a systems architect may not know what types of components will be used for the design implementation – design development focuses on selection of components, and their preliminary position and connectivity to other components. System structures are created through the decomposition and clustering of spaces, followed by the progressive specification of geometric details. The symbolic layout level focuses on room contours, connected symbolic wall segments, and assignment of properties to regions. Simple geometry corresponds to thick walls, fleshed-out columns, cut-out doors and windows. System behavior is enabled by the ability of the building occupants to function – the latter emanates from two sources: (1) functionality enabled by spaces and access to spaces, and (2) networked services (e.g, electrical, environmental micro-climates, security, etc.) integrated into the architectural domain. While many of these issues can be resolved with approximate/imprecise models of the final components to be used [15], it is important to note that few opportunities exist to test the final product prior to its full implementation. Therefore, formal mechanisms that will enable early validation of designer intent and design rule checking can vastly improve the quality and reliability of the building system prior to deployment.

Commercial tools such as AutoCAD [38], 3D Home Architect and TurboCAD Professional [8, 46] focus on the editing and presentation of Architectural/Mechanical CAD models/plans as blueprint-like drawings, 2-D designer viewpoints, and 3-D photorealistc renderings. Medium-end versions include support for pre-defined domain-specific features (e.g., architectural symbols),

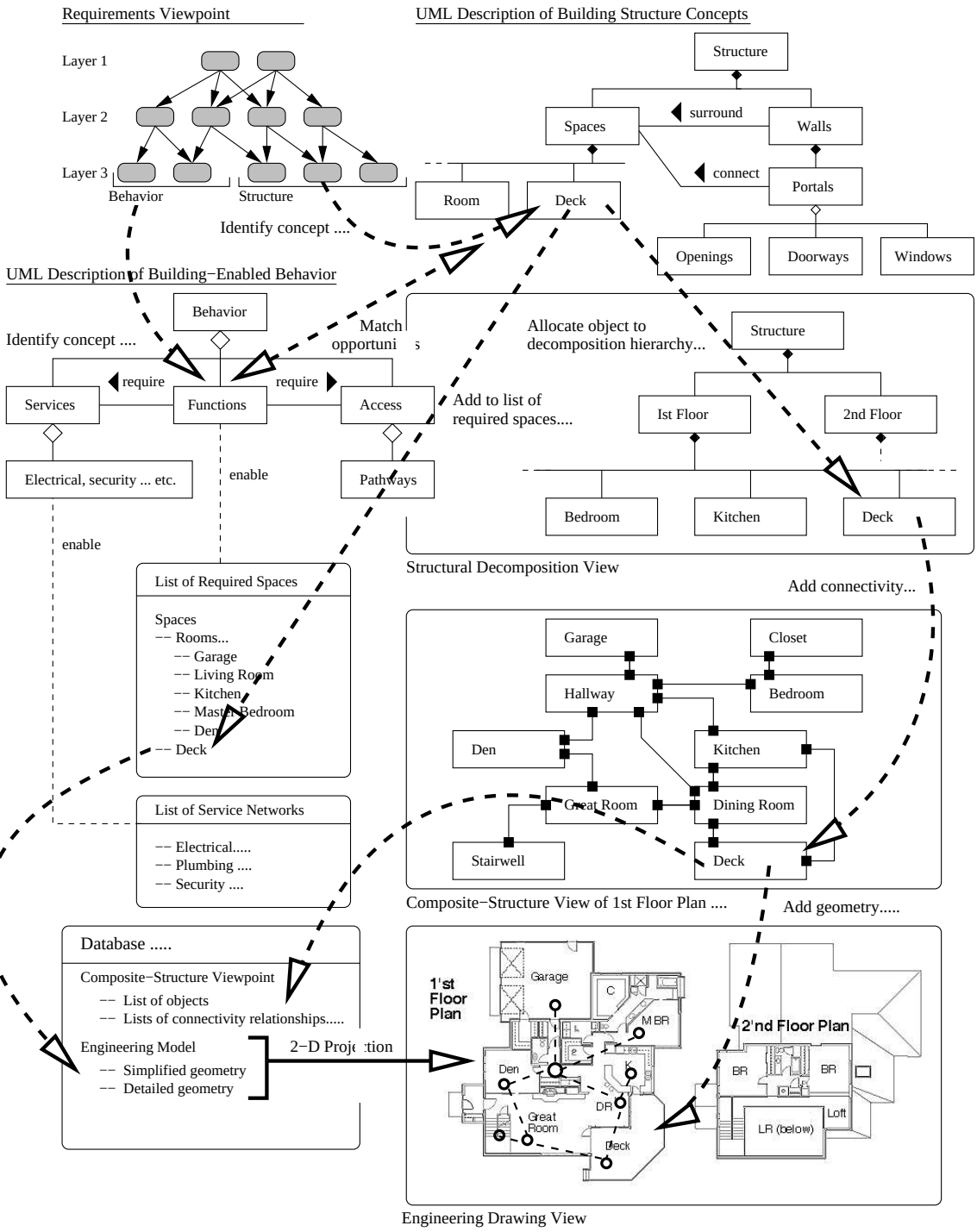



Figure A.3: Pathways from Requirements to UML Representations of System Behavior/Structure Concepts to Multi-Level Representations of Building Architecture. (Source: Austin [1])

dynamic dimensioning, basic solid and surface modeling (i.e., boolean operations and slicing), collision detection, and cost estimation. High-end versions go even further, including support for sophisticated solid modeling, management of constrained design dependencies (d-cubed constraint technology), multi-monitor visualization and export to standard interchange formats (e.g., STEP).

**Shortcomings of Present-Day Tools.** From a systems engineering perspective, present-day tools for architectural development are limited in the following ways:

1. Support for separation of design concerns (e.g., from the beginning, topology/connectivity concerns are connected to geometry concerns) is weak.
2. There is a lack of comprehensive support for spatial reasoning. As such, the tools are not easily extensible to layers of services.
3. Support for traceability of requirements to the engineering system itself is nonexistent.

In defense of item 2, research tools have been created for the exchange of data/information associated with symbolic building design representations [18, 53] and to evaluate whether a building floor plan adheres to certain requirements and standards [11]. Still, support for requirements traceability is completely missing.

**Traceability from Requirements to Building Design Models.** Figure A.3 shows a potential pathway from requirements to UML representations of system behavior and structure, to multi-level representations of buildings in a representation familiar to architects. UML can play a central role in the representation and visualization of intermediate products (i.e., application of "principles and practices" employed by professionals). For example, UML class and object diagrams are an ideal representation for: (1) Concepts (and relationships among concepts) associated with a particular problem domain or design concern, and (2) Organization (clustering and decomposition) of spaces into hierarchies.

The dashed arrows connecting requirements to UML classes are established by asking the question: What concept will be applied to satisfy this requirement? Then once that link is established, the continuing link to the engineering model is easily established – it is, after all, just the object instantiation of concepts modeled in the UML class diagram. On the back end, each class will contain attributes and methods needed to quantitatively evaluate object instances. Some

of this information may not be explicitly visible (e.g., exact square footage of a region, function of room, designated occupant, etc.). This extra information should be readily retrievable with a simple mouse or menu action. Research is needed to better understand the extent to which various types of constraints are supported by the UML class viewpoint.

System-level design alternatives are created by linking models of system-level behavior to the high-level structure, and imposing constraints on performance and operation (e.g., control logic; temporal logic; spatial logic). Floor planning processes need to adhere to three types of constraints: (1) topological (i.e., orientation, traffic/pathway, and location/adjacency concerns), (2) dimensional (i.e., size and space concerns) and (3) functional (e.g., aesthetic concerns) [10]. If the required functionality at lower levels of development cannot be satisfied (perhaps because the constraint values are too stringent), then the verification process will fail and the high-level developments will need to be adjusted to accommodate the demands of the lower level requirements (e.g., perhaps a space would need to be increased in size). The factoring process is guided by functionality that the region is expected to provide, and restricted by topological/geometric constraints [34, 38].

The heavy dashed arrows in Figure A.3 represent traceability links connecting requirements to specific system-level design concepts, which, in turn, will be implemented as entities in a building architecture object-model. Looking forward, designers should be able to click on a requirement and trace its implementation through the concept, structural decomposition, composite-structure and engineering drawing models. Conversely, designers should be able to click on an object (or group of objects) in a drawing and trace its existence back to a specific requirement (or groups of requirements). In this scenario a drawing is a detailed two-dimensional projection of an engineering model.

## A.2 The Semantic Web

In his original vision for the World Wide Web, Tim Berners-Lee described two key objectives [7]: (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 [27] 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 [23]. 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). In our view, future generations of computer support for storage, exchange, management, and visualization of requirements will make increasing use of Semantic Web technologies.

**Technical Infrastructure.** Figure A.4 illustrates the technical infrastructure that will support the Semantic Web vision. Each new layer builds on the layers of technology below it. The bottom layer is constructed of Universal Resource Identifiers (URI) and Unicode. URIs are a generalized mechanism for specifying a unique address for an item. They provide the basis for linking information on the Internet. Unicode is the 16-bit extension of ASCII text – it assigns a unique platform-independent and language-independent number to every character, thereby allowing any language to be represented on any platform.

The eXtensible Markup Language (XML) provides the fundamental layer for representation and management of data on the Web. XML grew out of demands to make the hypertext markup language (HTML) more flexible. The technology itself has two aspects. It is an open standard which describes how to declare and use simple tree-based data structures within a plain text file (human readable format). XML is a meta-language (or set of rules) for defining domain- or industry-specific markup languages. One well known example is the mathematical language specification (MathML), which captures the structure and content of mathematical notation [41]. A second example is the scalable vector graphics (SVG) markup language, which defines two-dimensional vector graphics

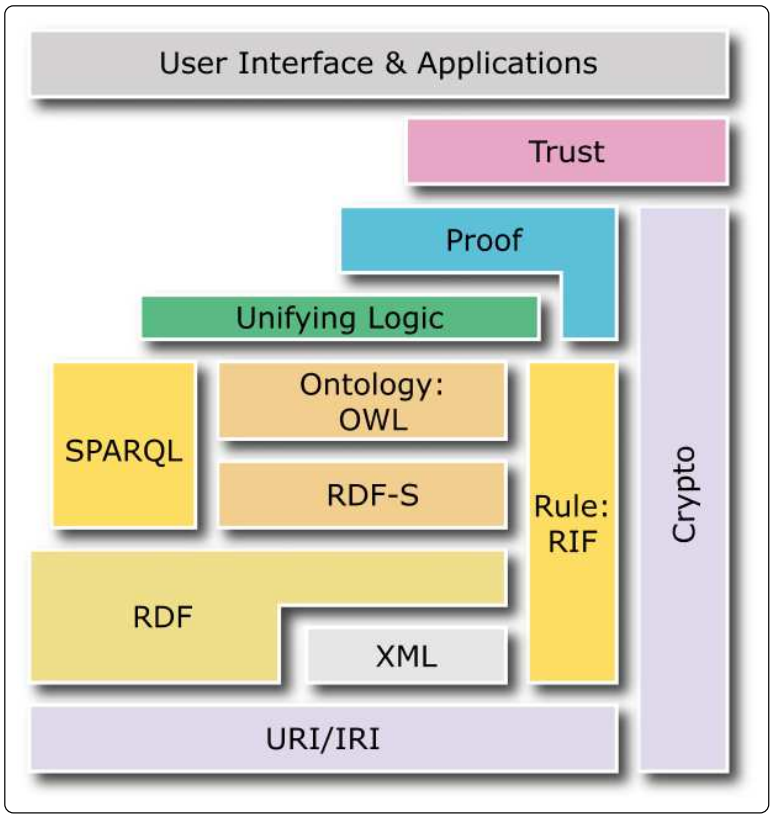



Figure A.4: Technologies in the Semantic Web Layer Cake

in a compact text format [57]. XML is being used in the implementation of AP233, a standard for exchange of systems engineering data among tools [44]. A key benefit in representing data in XML is that we can filter, sort and re-purpose the data for different devices using the Extensible Stylesheet Language Transformation (XSLT) [59, 67]. Stylesheets contain collections of rules and instructions that inform the XSLT processor how to produce the details of output. For example, a single XML file can be presented to the web and paper through two different style sheets.

**Limitations of XML. Need for the RDF Layer.** While XML provides support for the portable encoding of data, it is limited to information that can organized within hierarchical relationships. As illustrated in Figure 1.1, a common engineering task is the synthesis information from multiple data sources. This can be a problematic situation for XML as a synthesized object may or may not fit into a hierarchical (tree) model. A graph, however, can, and thus we introduce the Resource Description Framework (RDF). RDF is a graph-based assertional data model for describing the relationships between objects and classes in a general but simple way. The primary uses of RDF

are to encode metadata – information such as the title, author, and subject – about Web resources, and to designate at least one understanding of a schema that is sharable and understandable. The graph-based nature of RDF means that it can can resolve circular references, an inherent problem of the hierarchical structure of XML. An assertion is the smallest expression of useful information. RDF captures assertions made in simple sentences by connecting a subject to an object and a verb. In practical terms, English statements are transformed into RDF triples consisting of a subject (this is the entity the statement is about), a predicate (this is the named attribute, or property, of the subject) and an object (the value of the named attribute). Subjects are denoted by a URI. Each property will have a specific meaning and may define its permitted values, the types of resources it can describe, and its relationship with other properties. Objects are denoted by a "string" or URI. The latter can be web resources such as requirements documents, other Web pages or, more generally, any resource that can be referenced using a URI (e.g., an application program or service program). 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) [63].

**Ontology, Logic, Proof and Trust Layers.** The ontology, logic, proof and trust layers introduce vocabularies, logical reasoning, establishment of consistency and correctness, and evidence of trustworthiness into the Semantic Web framework. The logic, proof and trust layers are beyond the scope of this report.