Regular Paper

System Modeling and Traceability Applications of the Higraph Formalism

Kevin Fogarty[1] and Mark Austin[2, 3, *]

[1]Science Applications International Corporation (SAIC), Columbia, MD 21046

[2]Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742

[3]Institute for Systems Research, University of Maryland, College Park, MD 20742

Received 27 September 2007; Revised 30 April 2008; Accepted 10 January 2008, after one or more revisions
Published online in Wiley InterScience (www.interscience.wiley.com)
DOI 10.1002/sys.20113

ABSTRACT

This paper examines the use of higraphs as a means of representing dependencies and relationships among multiple aspects of system development models (e.g., requirements, hardware, software, testing concerns). We show how some well-known diagram types in UML have counterpart higraph representations, how these models incorporate hierarchy and orthogonality, and how each model can be connected to the others in a useful (and formal) manner. Present-day visual modeling languages such as UML and SysML do not readily support: (1) the traceability mechanisms required for the tracking of requirements changes and (2) built-in support for systems validation. Higraphs also deviate from UML and SysML in their ability to model requirements, rules, and domain knowledge relevant to the development of models for system behavior and system structure. To accommodate these demands, an extension to the basic mathematical definition of higraphs is proposed. Capabilities of the extended higraph model are examined through the model development for an office network computing system. © 2008 Wiley Periodicals, Inc. Syst Eng

Key words zaq;1

1. PROBLEM STATEMENT

Systems modeling is a fundamental component of the Systems Engineering process. Good modeling techniques allow for the comprehensive representation, organization, design, and evaluation of a system, from

*Author to whom all correspondence should be addressed (e-mail: austin@isr.umd.edu; kevin.p.fogarty@saic.com).

requirements, to structure, behavior, and beyond. Engineers are motivated to learn and use system modeling techniques in the belief that they enable and improve communication and coordination among stakeholders, thereby maximizing the likelihood of the right system being built correctly on the first try. Indeed, with a complete and correct system model in hand, ideally, implementation should be as simple as building the system per the model blueprint, which, in turn, is often represented through the use of system modeling languages. However, this approach assumes and requires two things: (1) that a system model exists that captures all facets of a system design (otherwise there will be holes in the "blueprint" where assumptions could be made) and (2) that the system model can be updated as requirements change, additional constraints are implemented, etc. In these two assumptions, we can find a point where the grand vision of system modeling and the reality of numerous present-day commercial engineering projects diverge. The problem is not that there are large flaws in current system modeling languages per se, but that existing system modeling languages (and associated model-driven methods) are relatively complex, and are difficult to use beyond the system modeling phase of the systems engineering lifecycle. In commercial settings, modeling languages in the form of popular commercial tools (see, for example, DOORS, Teamcenter (formally SLATE) and Visio [Microsoft, 2003; Teamcenter, 2008; Telelogic 2006]) are often forced into use by management on engineering projects. Too often personnel without true systems engineering skills are relied upon to use these tools, blindly, to create system models. If the underlying tools are implemented as islands of automation (or semiautomation) and are not connected together in a way that allows for flows of data/information among tools, then there is no automated way to create a trace from a requirement, to a component, to a behavior, to a test case. Support for change management is also weak due to the lack of a complete unified system model [Bell, 2004].

1.1. Scope and Objectives

The hypothesis of our work is that these modeling limitations can be mitigated through the use of higraphs, a topovisual formalism introduced by David Harel [Harel, 1987, 1988]. To date, the higraph formalism has been applied to a wide range of applications including statecharts in UML (Unified Modeling Language) [UML, 2003], expression of relationships in drawings [SysML, 2005] and urban forms [Dupagne and Teller, 1998], formal specifications in software development [Paige, 1995; Ramaswamy and Sarkar, 1997], component-based development of web applications [Wissen and Ziegler, 2003], and verification procedures in rule-based expert systems [Ramaswamy and Sarkar, 1997]. Higraphs have also made their way into Headway Software's reView, a tool for management of large software code-bases (the source code, libraries, packages, etc.) [Headway, 2001]. The common thread among these higraph-based applications is the use of nodes to represent allowable system states and edges to represent transitions between states (system functions) and/or dependencies between states or viewpoints. Hierarchies can be shown through enclosure; concurrent activities can be shown through orthogonality relationships. Because system models require and exhibit many of these same characteristics (states, transitions, hierarchies, concurrency), we surmise that higraphs might be a suitable abstraction for representing dependencies and relationships among multiple aspects of systems development models (e.g., hardware, software, electrical, mechanical concerns). Indeed, it is our contention that higraph representations can complement, and perhaps even coexist, with present-day UML and SysML representations of systems.

This paper begins with a detailed introduction to the mathematical formalities of higraphs and directed acyclic graphs. Section 3 focuses on existing visual modeling languages, and examines the goals, strengths, and weaknesses of the Unified Modeling Language (UML) [UML Forum, 2003] and the Systems Modeling Language (SysML) [SysML, 2005a, 2005b]. Section 4 covers the use of higraphs as a modeling tool for system requirements, system structure, and system behavior. We show: (1) how some well-known diagram types in UML have counterpart higraph representations, (2) how these models incorporate hierarchy and orthogonality, and (3) how each model can be connected to the others in a useful (and formal) manner. To accommodate these demands, the basic mathematical definition of higraphs is extended in Section 5. Finally, in Section 6 capabilities of the extended higraph model are examined through model development for an office network computing system.

2. INTRODUCTION TO HIGRAPHS

2.1. Definition of Higraphs

A higraph is a mathematical graph extended to include notions of depth and orthogonality. In other words [Grossman and Harel, 1997],

$$\textit{Higraph} = \textit{Graph} + \textit{Depth} + \textit{Orthogonality}. \quad (1)$$

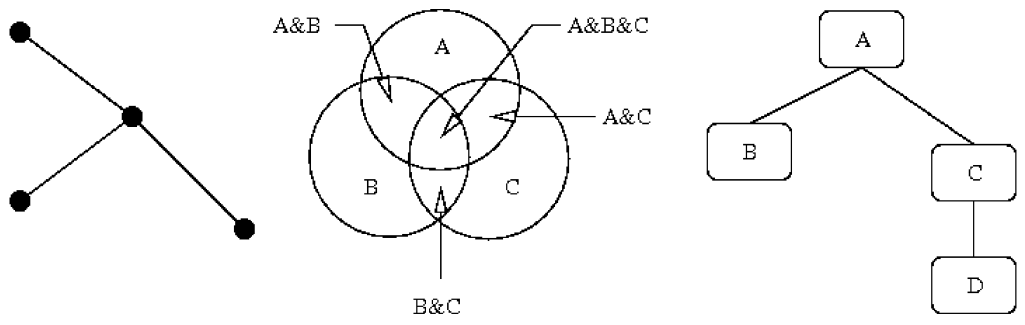


Figure 1. Fundamental elements in the definition of higraphs (left figure—basic graph structure; center figure—venn diagram; right figure—graph with blobs).

We denote the term “graph” by $G(V, E)$, where V is a set of vertices and E is a set of edges. The edges have no points in common except those contained in V. A directed graph is one in which the edges have direction—directed edges are called arcs (e.g., transitions in statechart diagrams). An edge sequence between vertices v_1 and v_2 is a finite set of adjacent and not necessarily distinct edges that are traversed in going from vertex v_1 to vertex v_2 [Chachra, Ghare, and Moore, 1979; Wiki, 2006]. The left-most schematic in Figure 1 shows, for example, a small mathematical graph that is generic in the sense that the nodes and edges have arbitrary meaning. All that is defined here is that four nodes and three edges make up this graph. The central node has some sort of relationship to the three other nodes through the edges. The term “depth” in Eq. (1) can be thought of as a defined hierarchy, and the term orthogonality can be thought of as a Cartesian product or partitioning. Orthogonal states provide a natural mechanism for modeling of systems that contain disjoint but concurrent subsystem developments and/or concurrent component behaviors. Higraphs also incorporate Euler Circles (or Venn Diagrams) to define the “enclosure, intersection, and exclusion” elements. Harel [1988] refers to these low-level atomic elements as blobs. The center schematic of Figure 1 shows, for example, a Venn diagram with relationships among three sets A, B, and C. Each set is define by its enclosures. Where set A and set B intersect, we see “A & B,” and this implies the exclusion of set C from this space. In the rightmost schematic of Figure 1, a graph structure is defined through connectivity relationships among the four blobs. Each blob has some sort of relationship (connectivity) to the central blob, Blob A.

2.2. Visualization of Relationships

A hierarchical relationship is defined by placing one blob inside another—see, for example, the left-hand schematic in Figure 2. Higraph edges represent relationships among system entities (e.g., physical connections, logical connections, and so forth). An edge can connect any node to any other node, even across hierarchies. The center schematic in Figure 2 shows, for example, an edge between blobs A and B, an edge from blob B to the node surrounding blobs C and D, and a single edge from the lower node (containing blobs C

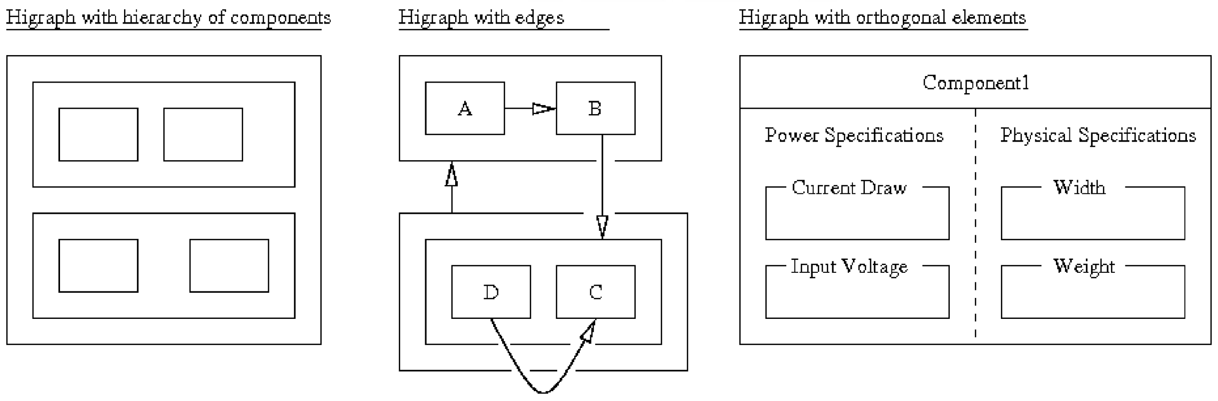


Figure 2. Visualization of hierarchy, orthogonality, and linking relationships in higraphs.

and *D*) to the upper node (containing blobs *A* and *B*). When the head of an edge (i.e., the arrowhead) terminates at anode, communication to all nodes and blobs within that node is implied. As we will soon see below, this notational mechanism allows for considerable simplification of complex systems.

Orthogonality concerns will be shown as a dashed line within a higraph. The right-most schematic of Figure 2 shows, for example, a team-based design where requirements are organized according to domain of expertise. Within Component1, two orthogonal regions (i.e., Power Specifications and Physical Specifications) are defined. Current draw, input voltage, width, and weight are all "lower level" specifications of the "higher level" Component1.

2.3. Mathematical Definition

The basic mathematical definition of a higraph can be summarized as follows [Grossman and Harel, 1997]:

- B is the set of blobs [nodes], b, that make up a higraph.
- E is the set of edges, e, that make up a higraph.
- ρ is the hierarchy function.
- Π is the orthogonality (or partitioning function).

The quadruple (B, E, ρ, Π) defines a higraph H. Harel [1987, 1988] provides the lowest level definitions of the hierarchy and partitioning functions. Applying these definitions to the higraph shown in Figure 3 yields the following equations:

1. $B = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$
2. $E = \{(i, h), (b, j), (l, c)\}$
   a. $e(l, c) = \{(l, f), (l, e)\}$
3. $\rho(a) = \{\{b \;\; B)\} \wedge \{b \text{ is defined from } a\} \wedge \{b \text{ is one level below } a\}\}$
   a. $\rho(a) = \{b, c, h, j\}$
   b. $\rho(b) = \{d, e\}$
   c. $\rho(c) = \{e, f\}$
   d. $\rho(g) = \{h, i\}$
   e. $\rho(j) = \{k, l, m, n, o\}$
   f. $\rho(d) = \rho(e) = \rho(f) = \rho(h) = \rho(k) = \rho(l) = \rho(m) = \rho(n) = \rho(o) = 0.$
4. $\Pi(H) = \cup \sum_{m=1}^{n} \pi_m \ (b \in B)$
   a. $\pi_1(a) = \{b, c, h\}$
   b. $\pi_2(a) = \{j\}$
   c. $\pi_1(j) = \{k, l, n\}$
   d. $\pi_2(j) = \{n, o\}$
   e. $\pi_1(b) = \pi_1(c) = \pi_1(d) = \pi_1(e) = \pi_1(f) = \pi_1(g) = \pi_1(h) = \pi_1(i) = \pi_1(k) = \pi_1(l) = \pi_1(m) = \pi_1(n) = \pi_1(o) = 0,$

where n is the number of orthogonal regions within an element selected from the set b. Higraphs are topovisual formalisms, meaning that nonmetric topological connectedness is important, as opposed to the size and

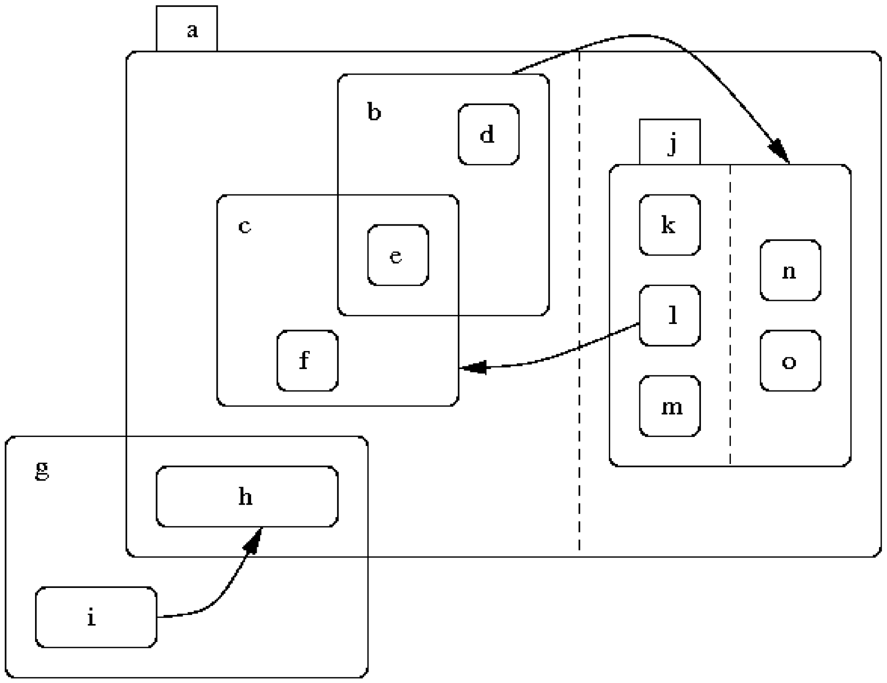


Figure 3. Example higraph for math modeling.

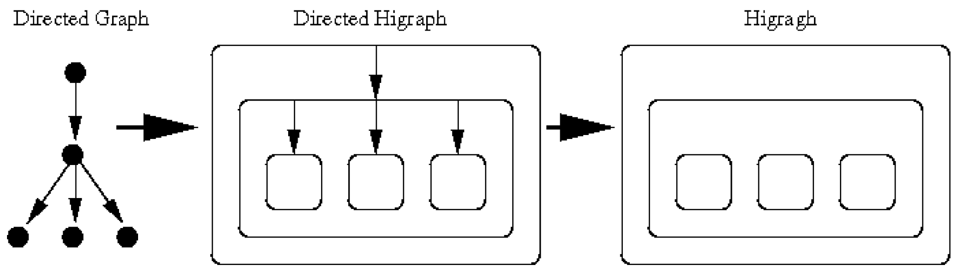


Figure 4. Step-by-step development of a higraph from a directed graph.

physical distance between nodes in a higraph [Munzner, 2000].

XOR Decomposition. Harel [1987, 1988] defines the concept of XOR decomposition as it relates to DAGs as: If *a* and *b* are nonintersecting and are contained in *c*, and *c* contains no other blobs, then *c* is the XOR of *a* and *b*—that is, to be "in" node *c* (to be in a state, have an attribute, etc.) means that you are "in" node *a* or node *b* (not both). And dually, if *a* and *b* intersect and their intersection contains blob *c* and none other, then *c* is also the XOR of *a* and *b*—that is, you are "in node *c* and *a*" or you are "in node *c* and *b*," but not all three. The appeal of XOR decomposition mechanisms to systems engineering is that they allow for top-down and bottom-up representation of systems organization.

Directed Acyclic Graphs and Higraphs. As defined by the National Institutes of Standards and Technology [Black, 2006zaq;2], a directed acyclic graph (DAG) is a directed graph with no path that starts and ends at the same vertex. Figure 4 shows a basic translation from DAG, to "Directed Higraph," to a higraph. Conversely, a DAG could be systematically derived from the higraph quadruple via appropriate algorithms (see Fig. 5). With a data structure such as a linked-list (with qualitative and quantitative attributes assigned to each node), the algorithms to trace through the system model already exist. As discussed later, these traces

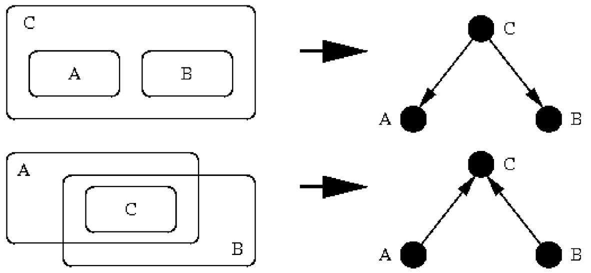


Figure 5. Systematic generation of DAG's from blob hierarchies.

through the system (the whole, unified system) are what provide real power to the higraph model.

A configuration is defined as the set of nodes corresponding to the vertices constituting a legal trace of the [DAG] higraph [Grossman and Harel, 1997]. The legal trace will be the result of some rule or command that causes the trace. In Figure 3 there are two valid traces that will return nodes *n* and *o*: ($A \rightarrow J \rightarrow N$) and ($A \rightarrow J \rightarrow O$). The command that executes this trace would be to find all components in the second orthogonal region of *j*. As we will soon see in much greater detail, by qualitatively or quantitatively defining *n*, *o*, *j*, and the meaning of the orthogonality in *j*, this trace will present the user with a unique view of the system.

2.4. Systems Engineering Application of Higraphs

For systems engineering applications the close relationship between higraphs and DAGs is important because, with the latter in place, appropriate rules (or search criteria) can be applied to traversals of the higraph structure to retrieve content. As illustrated in Figure 5, XOR decomposition allows for top-down and bottom-up representation of systems organization. With respect to visualization concerns, the hierarchical nature of higraphs allows for higher or lower levels of detail to be shown as needed. Moreover, by virtue of the many types of edges allowed in the higraph formalism (e.g., requirement assignment, allocation of behavior, complies with, satisfies, etc.), systematic tracing of the higraph edges will reveal much information about the validity of the system design. For instance, edge inspection will ensure the following:

1. All requirements (requirement nodes) can be traced to a system structure node (system component) or system behavior node (system behavior/function). If gaps exist, some requirements may not be met by the current system design.
2. All system behavior nodes (system behaviors/functions) can be traced to a system structure node (system component). This ensures correct

functional allocation; all behaviors are allocated to a specific component function.

3. No system structure or behavior nodes exist that cannot be traced to a requirement, thereby eliminating "gold plating," or the inclusion of components or capabilities not required by the specification.
4. The system structure is an instance of the domain structure (for normal, noninnovative, systems). This ensures that what you will build is in line with existing principles (e.g., physical laws). Likewise, ensure system behaviors comply with domain behaviors.

3. RELATED SYSTEM MODELING LANGUAGES

3.1. Capabilities and Strengths of UML and SysML

The goals of the Unified Modeling Language (UML) and the System Modeling Language (SysML) are to provide users with a ready-to-use, expressive visual modeling language (notation) so they can describe and exchange meaningful models [Rational, 1997]. Most engineers use UML informally—that is, diagrams are sketched as abstractions of a system description. Semi-informal uses of UML aim to create a one-to-one correspondence between UML and the system being described.

UML has evolved through two major revisions since the mid-1990s. UML 2, formalized in 2005, is defined by the list of diagrams shown in the upper half of Table I. Use case diagrams express required system functionality. Class diagrams express relationships among components in the system structure. Statechart and activity diagrams show two viewpoints of system behaviors. The remaining four diagrams summarize the mapping of behavior fragments onto structure, and details of their implementation. By adding communications, timing, and interaction overview diagrams, UML 2 makes significant improvements to the ways in which flows of information can be documented. The new Parts, Ports, and Connectors allow for a decomposition of systems into subsystems, components, parts, and so forth. This hierarchical representation is crucial to the modeling and evaluation of real-life systems [UML Forum, 2006].

SysML builds upon Versions 1 and 2 of UML, aiming to provide a visual notation for the development and evaluation of systems composed of both hardware and software. Development on Systems Modeling Language (SysML) began in 2003, and in 2005 the alpha spec was published [SysML, 2005b]. SysML supports

Table I. Types of Diagrams in UML2 and SysML: (1) Structure and Behavior Diagrams in UML2 and (2) Structure, Behavior, and Cross-Cutting Diagrams in SysML

Part 1. Diagrams in UML2	
Structure Diagrams	**Behavior Diagrams**
Class Diagram	Activity Diagram
Component Diagram	Use Case Diagram
Object Diagram	State Machine Diagram
Composite Structure Diagram	Interaction Diagrams
Package Diagram	• Sequence Diagram
Deployment Diagram	• Communication Diagram
	• Timing Diagram
	• Interaction Overview Diagram
Part 2. Diagrams in SysML	
Structure Diagrams	**Behavior Diagrams**
Block Diagrams	Activity Diagram (extends UML Activity Diagram)
• Block Definition Diagram (extends UML Class Diagram)	Use Case Diagram
• Internal Block Diagram (extends UML Composite Structure Diagram)	State Machine Diagram
Parametric Constraint Diagrams	Sequence Diagram
• Parametric Definition Diagram	
• Parametric Use Diagram	
Cross-Cutting Diagrams	
Allocation Diagram	
Package Diagram (extends UML Package Diagram)	
Requirements Diagram	

the specification, analysis, design, verification, and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities [SysML, 2005b]. As shown in the lower half of Table I, the SysML diagram types are organized into three sections; diagrams for modeling system structure, for modeling system behavior, and those that cut across viewpoints. The new parametric diagram follows the graphical conventions of a UML internal structure diagram showing a collaboration [SysML, 2005b]. Parametric constraints can be used in tradeoff studies to show what happens to one (internal) characteristic of a block, when characteristics in another block are changed. Cross-cutting diagrams get their name from the nature of the information contained in each—in other words, these diagrams show how a particular concern (requirement) cuts across the structural and behavioral domains. Compared to UML, SysML offers the following new features:

1. **Block Stereotypes.** The SysML Block Stereo type is based on the UML concept of composite structures. Blocks can have internal features (attributes, operations) and can own ports. The extension of UML ports in SysML as flowports provides a far more complete system model in which blocks can be connected (physically and/or logically) to other blocks.
2. **Allocations.** SysML extends the UML trace comment with their new allocation property. Functional allocation is the assignment of functions (requirements, specifications, behaviors, etc.) to system components. Support for functional allocations is needed especially in the development of larger systems where design and implementation may not occur at the same place or time. UML versions 1 and 2 make little reference to functional allocation (aside from swimlanes in an Activity diagram).
3. **Requirements Modeling.** SysML provides modeling constructs to represent requirements and relate them to other modeling [system] elements [SysML,2005a]. SysML introduces an actual requirements node which contains information about requirements such as identifier, text, source, and method of verification. These requirements nodes can be used in Block Definition Diagrams (SysML version of a UML class diagram) to show a hierarchy of requirements. Requirements can also be mapped to other elements by derivation, verification, and satisfaction paths (e.g., a diagram can show how a specific requirement is assigned to a component in the system structure.)

3.2. Weaknesses of UML and SysML

The following quote from Berkenkotter [Berkenkotter, 2003] captures perhaps the most significant weakness of UML: "One of the most frequently discussed weaknesses of UML 1.4 is its usability as it consists of an overwhelming number of diagrams and elements. While diagrams may represent different views on a system, there is no mechanism to define the interconnections or dependencies among the diagrams describing a system." In other words, there are too many places to capture information (in the large number of available diagrams), and too few ways to show relationships between the diagrams. This has not changed with UML 2. From a systems engineering perspective, little effort is given to requirements modeling, functional allocation and domain specific (customized) viewpoints. To be fair, this is done in part, to keep the focus of UML remaining on software and real-time software systems.

UML 2 provides little support for requirements definition and traceability. In an effort to mitigate this deficiency, Letelier [2002] documents an entire requirements traceability meta-model. This meta-model works within the specifications of UML to show not only requirements traceability, but also traceability throughout the rest of the system. This contribution is important because it highlights the lack of support in UML for functional allocation at a system level. Letelier also extends UML to include an "assignedTo" stereotype, which can be used in Requirements Allocation activities (assigning a requirement to a component or behavior) within a UML model.

While SysML makes significant improvements on UML in terms of modeling traditional systems engineering processes, there are a few areas of weakness in the SysML alpha release:

1. **Weak Support for Diagram Connectivity.** Something that is not addressed in the SysML specification is the idea of interconnections between diagrams. SysML is much better than UML at showing multiple ideas on a single diagram (i.e., a component in a structure diagram with its parent requirement tag and test case tag). However, an alternative and potentially better implementation would allow links from a requirements diagram to a structure diagram—instead of manually placing a <<requirements>> comment in a structure diagram. By allowing links between diagrams, as a higraph model allows, you would minimize the total number of

complete diagrams, but could keep any number of relations.

2. **Weak Support for Allocations.** As discussed earlier, there is a strong effort to model allocations in SysML. However, while the notion is fundamentally correct (as documented in the SysML specification), there seems to be no rules on allocations. In other words, how do we know if the <<allocate>> tag is correct? Although there always must be reliance on the human creating model, under this specification, an engineer could conceivably allocate a behavior to a requirement (instead of allocating the requirement to a behavior), or allocate five behaviors to a Block (structure) that does not have sufficient attributes or functions to support those behaviors.
3. **Weak Support for Hierarchy among Allocations.** To complicate matters, while SysML specifies hierarchical relationships among structure, behaviors (blackbox versus whitebox), and requirements, there is no clear definition of hierarchy among allocations. For instance, requirements can be allocated to subcomponents, but it is not clear how those allocations are dealt with if there is a change to a higher-level component. This may have been overlooked because, in software, inheritance and encapsulation mechanisms can be relied upon to propagate changes from a class to its lower-level subclasses. However, in other engineering applications (i.e., physical integrations) there needs to be a way in the model to ensure that when the dimensions of a physical component changes (high level change), the dimensions of subcomponents stay within specification (leads to low level change).
4. **Weak Mathematical Foundation of UML/SysML.** UML and SysML are both defined via their meta-models, that is, a meta-model for what kinds of diagrams will be supported, and the features within each type of diagram. The meta-model is enough information for computer vendors to: (1) implement software that will support the construction of diagrams to describe engineering systems (e.g., Microsoft Visio, Rational Rose) and (2) develop languages for the exchange of UML/SysML data/information among tools (e.g., XMI and AP233) [Buller, 2003zaq;2; Oliver, 2002]. The principal problem with meta-models, versus a mathematical foundation, is that the former provides only weak enforcement of relationships among system entities. As a result, software tools like Microsoft Visio, allow a systems engineer to create UML diagrams that don't make any sense with respect to real-world entities.

Higraph models have the benefit of being defined by a mathematical formula, thereby ensuring that all relations between requirements, structure, and behavior entities are formalized. These relationships must be honored for the model to be valid. We also assert that by forcing directional allocations (i.e., requirements to components, behaviors to components) to the lowest level possible, not only will clarity of decision making in systems engineering be improved, but it will also allow for early validation of system correctness. System design rules could be created that only allow certain types of edges (e.g., allocations) to connect a requirement to a behavior, or connect a behavior to a function in a system structure component. The follow-up enforcement of rules for allocations (edge connectivity in higraphs) provides a basis for traceability-enabled error checking within a system model. For example, all edges could be examined to ensure their end-points are compatible (e.g., a requirement to a component attribute, a behavior to a component function) and complete (e.g., all requirements have edges to either a behavior or function).

4. SYSTEMS ENGINEERING MODELING WITH HIGRAPHS

Now that we have examined existing system modeling languages, and proposed ways for improvement through the use of higraphs, we will show how the higraph formalism can be applied to the representation and organization of system modeling entities (i.e., requirements, structure, and behavior), and the traditional diagrams that describe them. Sections 4.1–4.4 follow the development process shown in Figure 6. System behavior/functionality is defined by use cases. Fragments of required behavior are defined using activity and sequence diagrams. Most of the requirements correspond to constraints on performance, interface, and economic concerns that an implementation would need to satisfy. Section 4.6, in particular, describes how higraphs can link components from higraphs together to produce flows of design information generated during the system development.

4.1. Use Case Modeling

Use case diagrams show what actions external users (e.g., users, operators, maintainers, etc.) can perform using the system. By replacing the stick-figure icons representing actors with the less aesthetically pleasing Higraph nodes, traditional use case diagrams can be

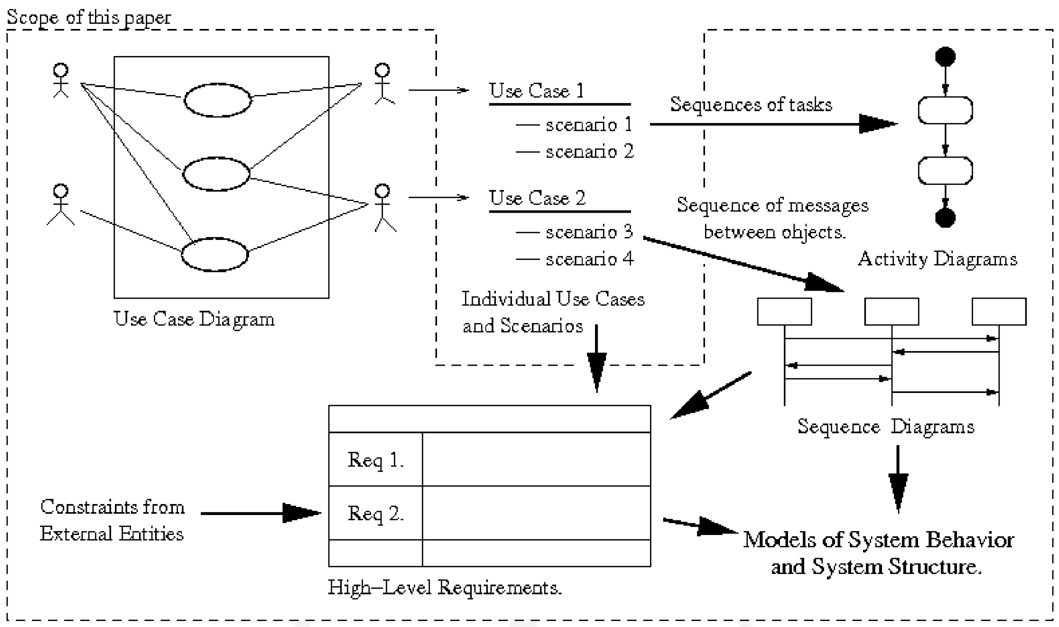


Figure 6. Pathway from operations concept to models of behavior/structure to requirements.

converted to higraph use case diagrams. See, for example, Figure 7.

4.2. Requirements Modeling

To model system requirements using higraphs, we will define how the graph elements can be used. The nodes in a requirements higraph will represent individual requirements (whatever the domain). All the node has to capture is the text of the requirement. The node (it may be best to think of a node as the instance of a class in an object oriented paradigm) could have as many text fields as necessary (e.g., number, textual description, priority, and owner/stakeholder). Multiple levels of requirements may be represented by a hierarchy of nodes. Various interpretations in the edges are possible—for example, "parent" and "child" requirements, high-level requirements and low level requirements, explicit requirements and derived requirements.

Requirements are commonly organized into tree (and graph) hierarchies, especially for team-based design [Austin, Mayank, and Shmunis, 2006]. But this is not the only possibility. Another logical organization of requirements is by domain. These domains may represent different types of requirements (e.g., physical specifications, electrical specifications, mechanical specifications), requirements from different stakeholders, or may represent requirements from outside of the technical realm (technical specifications, project cost requirements, project schedule requirements, project

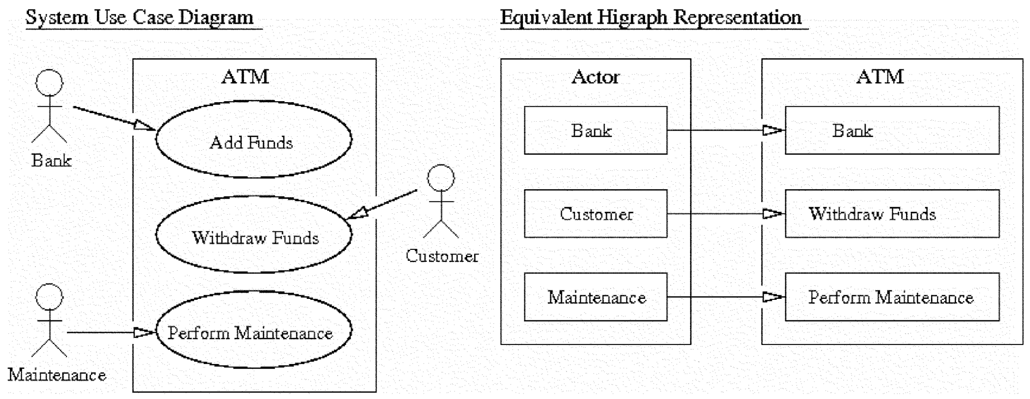


Figure 7. ATM use case diagram and counterpart higraph representation.

staffing requirements). Sometimes domain organization will overlap, for example, when requirements are common to multiple domains and/or they represent the interface between domains. Introducing orthogonality to the requirements higraph allows for the logical and visual separation of requirements from different domains.

Orthogonality is a feature of higraphs that can be used to define, separate, and logically group domain requirements. Consider, an example, where power and physical requirements are organized into a higraph, as shown in the upper half of Figure 8. Required electrical performance of the engineering system is covered with three requirements. Requirements 1.0 and 2.0 are explicit requirements; requirement 1.1 is derived from requirement 1.0. Notice how the hierarchy of requirements is implied without the use of edges. The corresponding DAG for this organization of requirements is shown in the lower half of Figure 8. When an orthogonality is shown in a DAG, the DAG takes on an "and/or" construct. The orthogonal relationship between Power Requirements and Physical Requirements is represented as an "or", but the other (hierarchical) relationships are shown as "ands." Harel creates this theory in Grossman and Harel [1997].

4.3. System Functionality and Behavior

Activity diagrams and sequence diagrams are both ideal mechanisms for visualizing fragments of system functionality. Activity diagrams, with their activities (nodes) and transitions (edges) can easily be modeled as a higraph. Decision elements are supported by a specific type of node. Parallel behaviors are supported by orthogonally divided activities. Figure 9 shows a simple example taken from the automobile domain. Likewise, sequence diagrams which show a sequence of events over time, can be modeled using higraphs. See, for

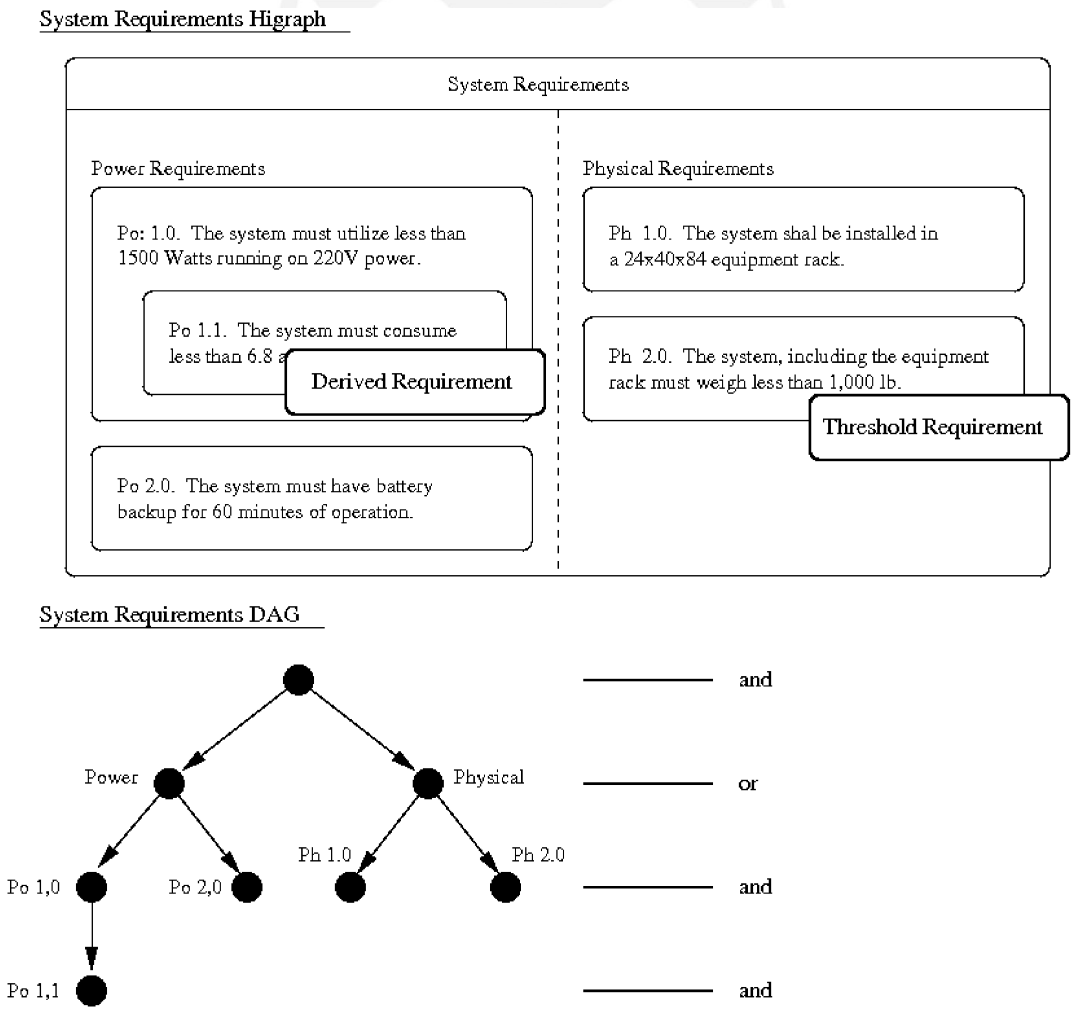

System Requirements Higraph
System Requirements
Power Requirements
Po: 1.0. The system must utilize less than 1500 Watts running on 220V power.
Po 1.1. The system must consume less than 6.8 a
Derived Requirement
Po 2.0. The system must have battery backup for 60 minutes of operation.
Physical Requirements
Ph 1.0. The system shal be installed in a 24x40x84 equipment rack.
Ph 2.0. The system, including the equipment rack must weigh less than 1,000 lb.
Threshold Requirement
System Requirements DAG
Power
Physical
Po 1,0
Po 2,0
Ph 1.0
Ph 2.0
Po 1,1
and
or
and
and

Figure 8. System requirements.

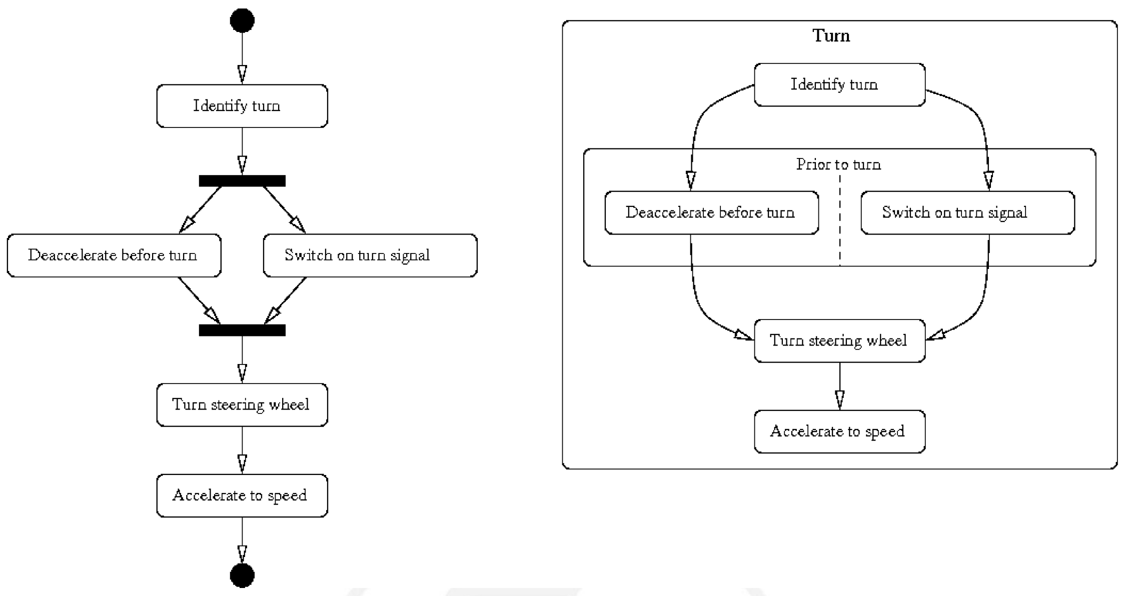


Figure 9. Activity diagram and equivalent higraph representation for turning a car.

example, Figure 10. To do this, we have adapted a concept described in Minas [1998]. Note that messages (edges) originate from traditional structure object nodes (driver, door, doorlock), but they must pass through a "time" node (with an attribute counting time) before arriving at another structure node.

Higraph Modeling of System Behavior. Detailed models of system behavior emanate from synthesis and organization (sequences, loops, hierarchies, concurrencies) of behavior fragments. By paying attention to the grouping of these states (represented by nodes or blobs), behavior models can remain in proportion to the size of the system structure model. Edges are events, internal or external, that cause the system to change states. Figure 11 shows, for example, three concurrent behaviors—transmission, heat, and lighting systems—

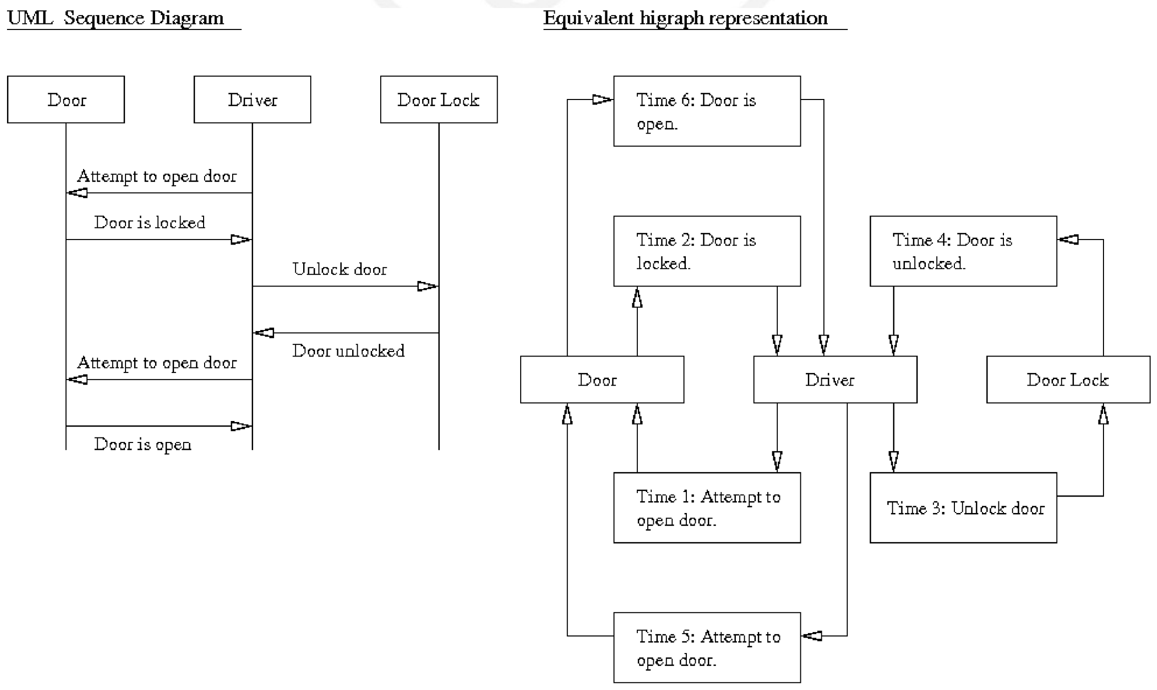


Figure 10. Sequence diagram and equivalent higraph representation for entering a car.

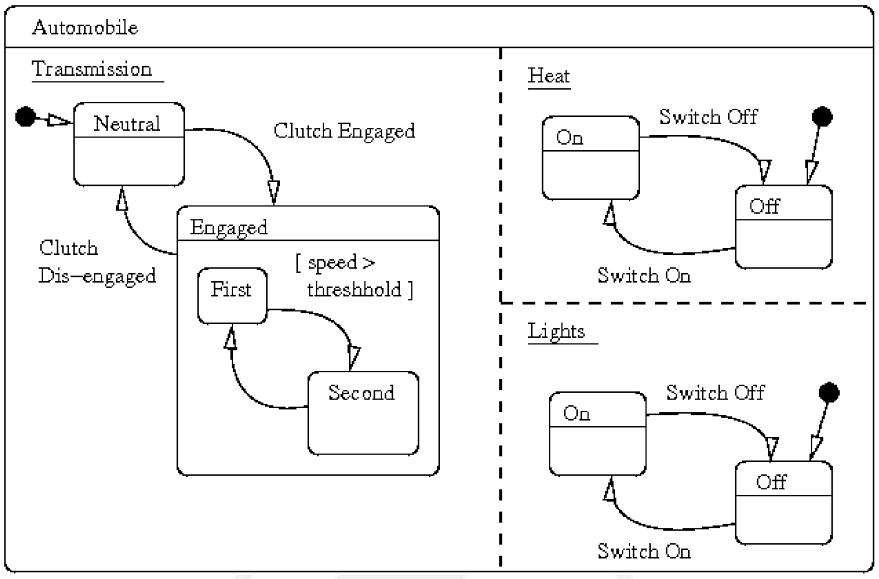


Figure 11. Higraph system behavior diagram automobile.

in a modern automobile. For the heat and lighting systems, only the top level of behavior is shown. The transmission system is presented with two levels of detail. Each orthogonal region has a distinct initial state. Changes in system state (e.g., the transmission moves from drive to neutral) are triggered by external events. Internal events correspond to behaviors defined within the nodes in the system structure model. External events should flow from use cases. Edges can also be labeled with values that are the result of behaviors that occur within a state (node). A single edge can represent a transition that affects any number of states. Since nodes (states) can be grouped so they share common edges (transitions), when a component is added to the system, it can be grouped so the total number of states does not increase exponentially.

4.4. System Structure (and System-Level Design)

By design, system structure modeling with higraphs is very similar to system structure modeling with UML and SysML. For both UML and SysML, the primary artifact of the system structure is the class diagram. UML class diagrams and SysML block diagrams show a hierarchy of classes/blocks, each with attributes and behaviors, and rules for assembly. The latter can involve composition, aggregation, multiplicity, and generalizations (among others). The classes/blocks, and their hierarchical arrangement, define the structure of a system.

In a higraph model of system structure, the nodes represent classes, attributes, and functions, and edges show association (or other general relationships) between classes. Attributes and behaviors are defined within class nodes. The hierarchical arrangement of nodes in a system structure diagram represents a class hierarchy and shows aggregation and composition relationships. Aggregation can be thought of as a weak "has-a" relationship between classes. The relationship is weak in the sense that when the parent class is deleted, the subclass(es) will still exist. Composition, on the other hand, is a strong "has-a" relationship where if the parent class is deleted, the subclass(es) will not exist. See UML Glossary [2006] for a complete UML Glossary. Orthogonal regions can separate classes that aggregate or compose a parent class. And, finally, in a higraph model of system structure, edges show generalization, representing an "is-a" relationship between classes. Inheritance of attributes and functions would follow these edges. The capability to formally model inheritance is significant due to the weak support for this in SysML and the notion that a system model, that itself could be modeled in an object-oriented fashion (to simplify a software implementation) would be desirable.

Example: UML Based Structure Model of an ATM. A key advantage of higraphs is the ease with which varying amounts of detail can be shown. Figure 12 shows three views of a system-level design for an ATM machine. The top-left schematic shows a top-level higraph representation for an ATM system structure composed of hardware and software classes. A detailed view of the attributes and functions for the hardware and software is shown on the top right-hand side. Additional details of the ATM software implementation (i.e., Customer Verification Software) are shown in the lower-most schematic. Nodes at the bottom of the diagram (at the end of the open, unidirectional arrow) are generalizations of the ATM Hardware parent class. As such, these child classes inherit all attributes and functions that exist in the parent class, yet may have their own unique attributes and functions.

For the purposes of comparison, counterpart UML diagrams are shown in Figures 13–15. Nodes at the ends of the open, unidirectional arrows are generalizations of the ATM Hardware parent class and ATM Software parent class. As such, these child classes inherit all attributes and functions that exist in the parent class, yet may have their own unique attributes and functions.

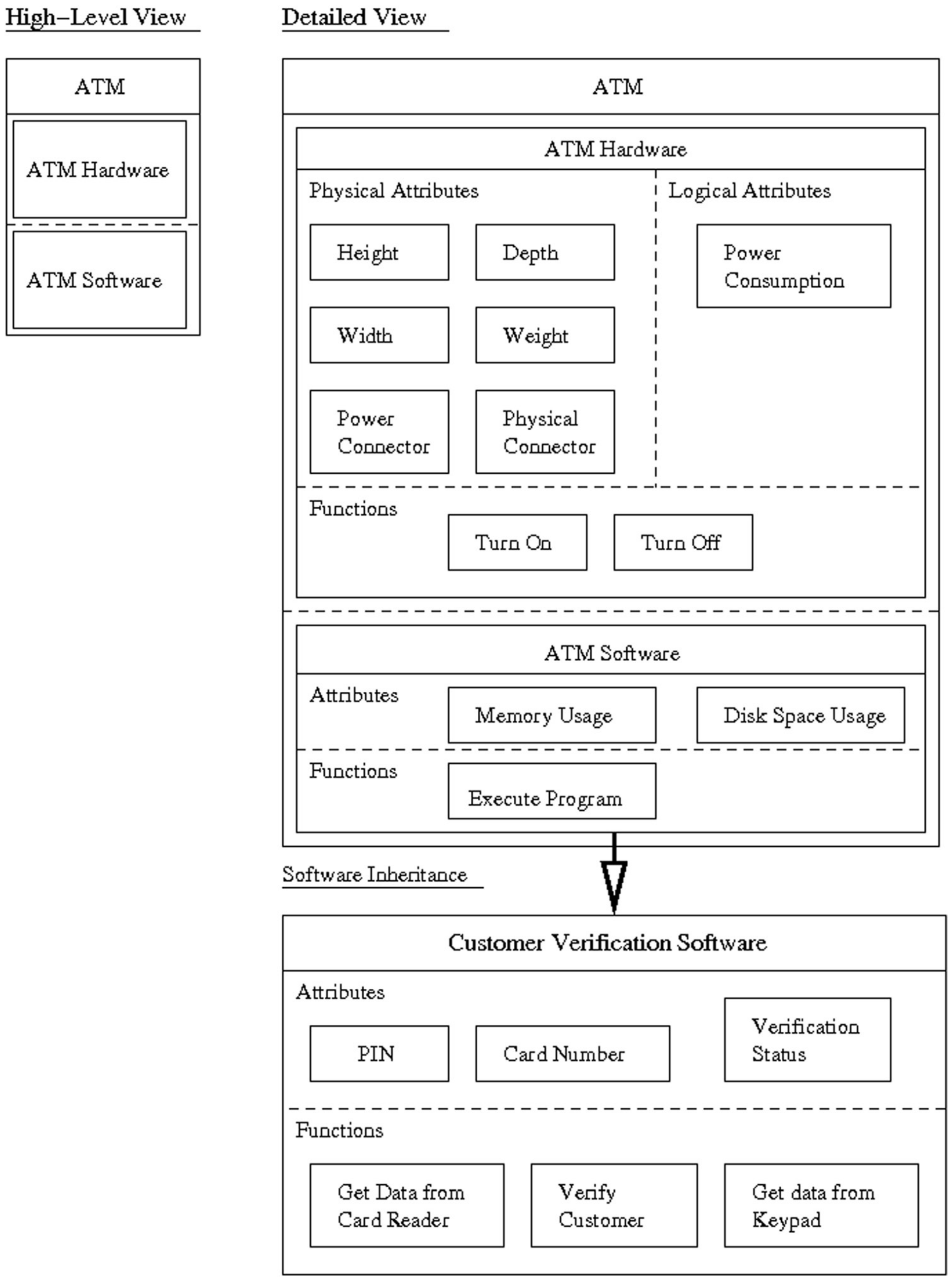


Figure 12. High-level, detailed, and software inheritance views of system structure and hardware/software system breakdown for an ATM Machine (adapted from Austin [2002]).

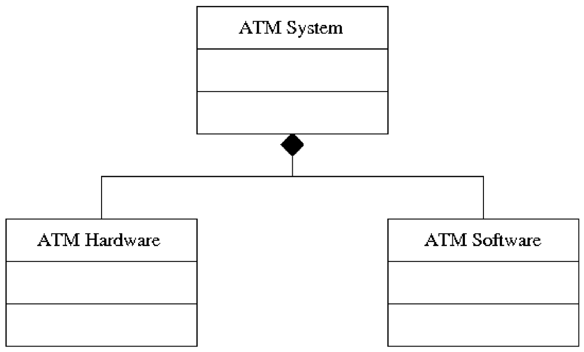


Figure 13. High-level UML class diagram for ATM machine.

Because the UML class diagram (and equivalent Higraph diagram) shows component attributes and functions, it can be thought of as a system design model—not just a system structure model. Individual attributes and functions are defined within individual nodes and are arranged hierarchically within the class to which they belong. Even within the attributes region of the ATM Hardware class, two orthogonal regions are shown. This represents physical and logical attributes, both of which compose the attributes for the ATM hardware class.

4.5. Modeling Domain Requirements, Structure and Behavior

While the modeling of domain rules is established and mature, to do so with a higraph representation is new and novel. Higraphs deviate from UML and SysML in their ability to model requirements, rules, and domain knowledge (e.g., relevant principles of science such as electromagnetic fields equations for a communications system) relevant to the development of models for system behavior and system structure.

4.6. System-Level Modeling and Connectivity

Because higraph models allow for arbitrary connections among elements, their primary strength lies in explicit support for traceability (via edges) between models of system requirements, system structure, and system behavior. Indeed, although each of aspects may be defined in their own higraph model, the formalism allows for their linking into one large higraph, thus creating a true system model.

Consider, for example, the high-level connectivity of requirements, structure, and behavior higraph models shown in Figure 16. The system requirements higraph model is partitioned into three orthogonal regions: one for physical requirements, a second for functional requirements, and a third for interface requirements. Since we have chosen to separate physical and functional requirements into different orthogonal regions (a logical separation in this case), we require an "interface" through which these requirements could connect to each other. By design, the interface requirements node spans between the physical requirements and functional requirements, and any edges would have

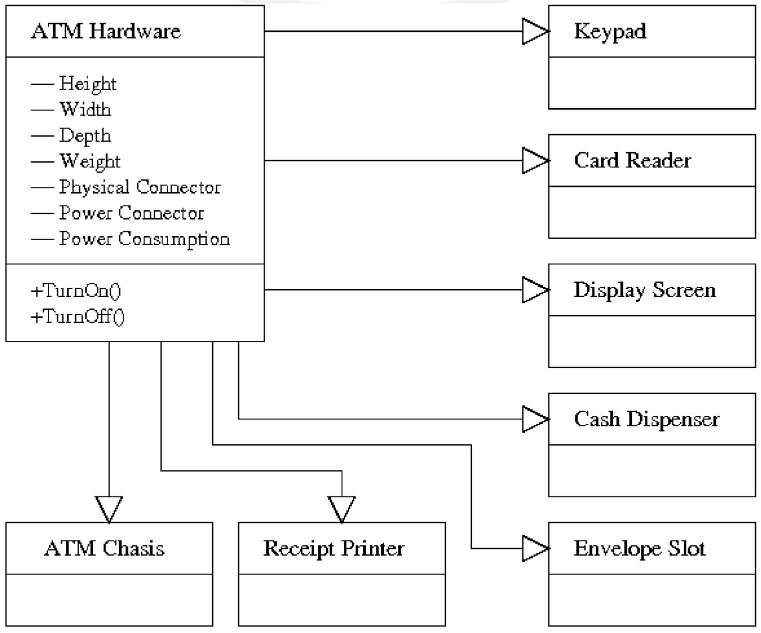


Figure 14. UML system structure diagram aggregation.

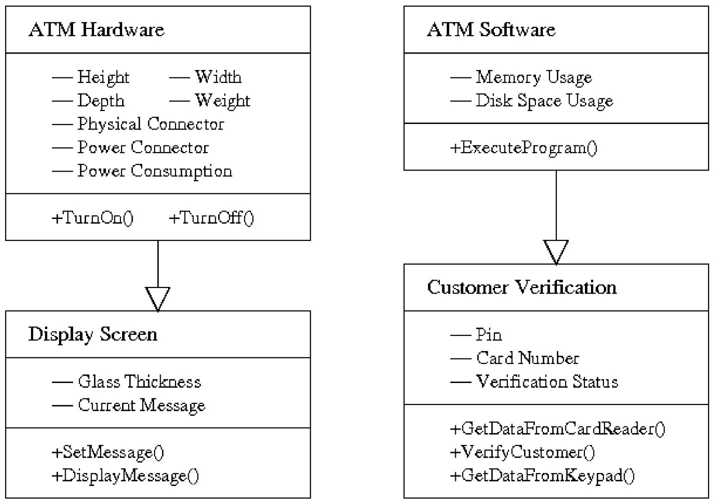


Figure 15. UML system structure diagrams for hardware and software components.

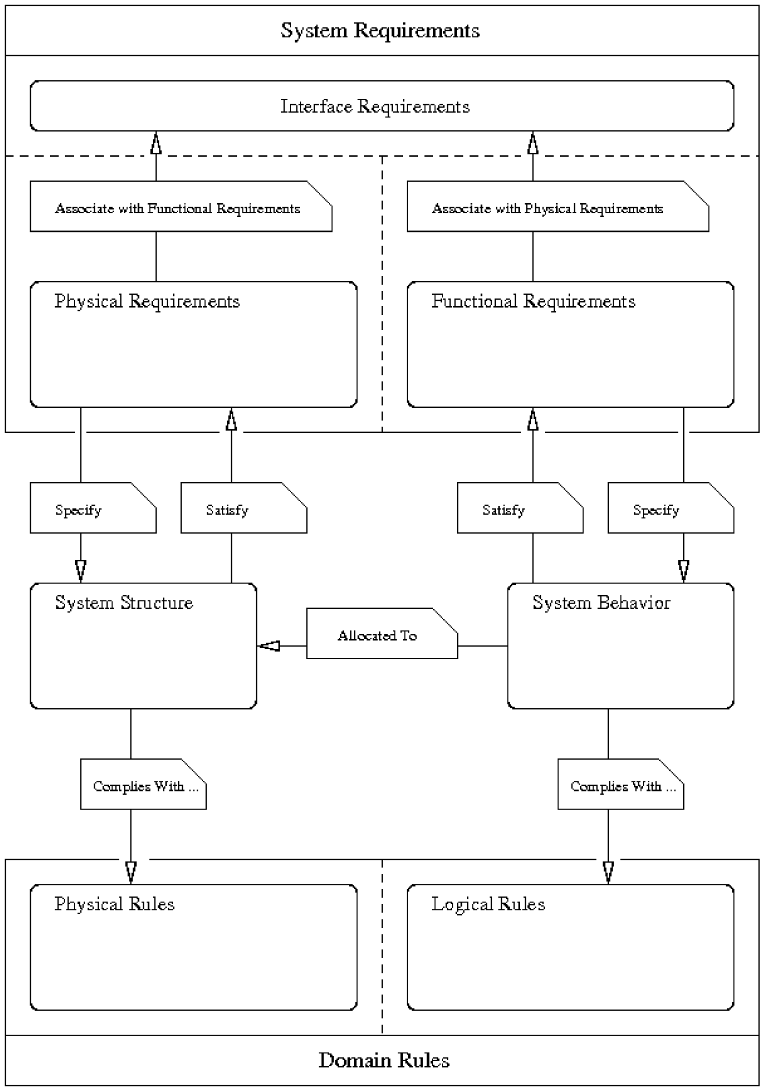


Figure 16. Higraph-based requirements, system model, and framework for domain rule checking.

to pass through a node in the interface requirements area to go from physical to functional (or vice versa). Edges connecting the three higraphs show what pieces of the system structure satisfy specific physical requirements, and what system behaviors satisfy specific functional requirements. Finally, the lower half of Figure 16 shows how models of system structure are linked to domain rules (physical realities), and how domain behaviors comply with domain rules (functional realities).

5. EXTENDED MATHEMATICAL AND LOGICAL MODELING

When Eq. (1) is applied to the higraph representation of an actual system, the result is a DAG for the system representation. From a systems engineering perspective, however, the formulation is missing specific details for how fragments of behavior and attributes of system structure map onto the DAG. Therefore, in this section, we extend Eq. (1) to include assignment of types to nodes and edges in higraphs, and definitions to hierarchies and orthogonalities. Table II contains a summary of the extended higraph element definitions.

Nomenclature. Let B and E be the sets of nodes and edges that make up a higraph. We will define B to be made up of B_1 (set of all system requirement nodes), B_2 (set of all system component nodes), and B_3 (set of all system behavior nodes). Lower level details are represented through extension of the subscript notation. For instance, B_2 may be defined as being made up of $B_{2\text{-}1}$ and $B_{2\text{-}2}$ (set of all system component attribute nodes, and set of all system component function nodes, respectively). So, $B = (B_1, B_2, B_3)$, where $B_2 = (B_{2\text{-}1}, B_{2\text{-}2})$.

Higraph edges may represent (but are not limited to) the following: (1) assignment of requirements, (2) assignment of a requirement to a component, (3) assignment of a requirement to a behavior, (4) assignment of a behavior to a component (functional allocation), (5) inheritance between system components, and (6) a transition from one system state to another, corresponding to a behavior. We will define E to be made up of E_1 (set of all requirement assignments), E_2 (set of all functional allocations), and E_3 (set of all behavior transitions). Further, E_1 may be defined as being made up of $E_{1\text{-}1}$ and $E_{1\text{-}2}$ (set of all requirements assigned to system components, and set of all requirements assigned to system behaviors, respectively). So, $E = (E_1, E_2, E_3)$, where $E_1 = (E_{1\text{-}1}, E_{1\text{-}2})$.

Hierarchy in higraphs might represent (but is not limited to) the following: (1) derived requirements, (2) system component specification, (3) allocation of an attributes to a component, (4) allocation of a function to a component, (5) high level or low level system behaviors. If ρ is the set of hierarchies that make up a higraph, we will define ρ to be made up of ρ_1 (set of all derived requirements), ρ_2 (set of all component specifications), and ρ_3 (varying levels of system behaviors). Further, ρ_2 may be defined as being made up of $\rho_{2\text{-}1}$ and $\rho_{2\text{-}2}$ (set of all requirements assigned to system components, and set of all requirements assigned to system behaviors, respectively). So, $\rho = (\rho_1, \rho_2, \rho_3)$, where $\rho_2 = (\rho_{2\text{-}1}, \rho_{2\text{-}2})$.

Orthogonality in higraphs may represent (but are not limited to) the following: (1) logical partitioning of requirements (e.g., physical requirements, functional requirements), (2) structural relationships (e.g., composition and aggregation), and (3) concurrent system be-

Table II. Summary of Higraph Element Definitions

	Requirements Model	Structure Model	Behavior Model
Nodes	• System Requirements	• Component Attributes • Component Functions	• System States
Edges	• Assignment of a Requirement to system component • Assignment of a Requirement to system behaviors	• Inheritance • Assignment of behavior to system component	• Transitions between states
Hierarchy	• Requirements Hierarchy	• Allocation of an attribute to a component • Allocation of a function to a component	• Behavior Hierarchy
Orthogonality	• Logical partition of requirements	• Composition • Aggregation	• Concurrent behavior

haviors. If Π is the set of orthogonalities that make up a higraph, we will define Π to be made up of Π_1 (set of requirement partitions), Π_2 (set of all structural relationships), and Π_3 (set of all concurrent system behaviors). Further, Π_1 may be defined as being made up of $\Pi_{1\text{-}1}$ and $\Pi_{1\text{-}2}$ (set of all physical requirements, and set of all functional requirements, respectively), and Π_2 may be defined as being made up of $\Pi_{2\text{-}1}$ and $\Pi_{2\text{-}2}$ (set of all composition relationships, and set of all aggregation relationships, respectively). So, $\Pi = (\Pi_1, \Pi_2, \Pi_3)$, where $\Pi_1 = (\Pi_{1\text{-}1}, \Pi_{1\text{-}2})$ and $\Pi_2 = (\Pi_{2\text{-}1}, \Pi_{2\text{-}2})$.

6. HIGRAPH MODELING OF AN OFFICE COMPUTING NETWORK

In this section capabilities of the extended higraph model are examined through model development of an office network computing system. To simplify the model development we assume that the network is already in place—it follows that system requirements and components will also be in place. The principal goals of the example are to demonstrate that the office computing network system can be represented in higraph form, which, in turn, can be used to respond to queries and changes to system requirements. The second important purpose is to demonstrate partial formulation of the math model from which pseudoqueries of the system higraph model can be performed. For details on higraph representations for individual requirements, and structure and behavior models, the interested reader is referred to Fogarty and Austin [2007].

6.1. Structural Requirements Traceability

We begin by showing examples of connectivity, via edges, for allocation of requirements to component attributes and system behaviors, allocation of system behaviors to component functions, and traces from domain requirements to system requirements. Figure 17 shows the association between the system's behavior requirements and the system use case higraph. Figure 18 shows the allocation of the system cost requirement to attributes in system structure components—system hardware components and system software components in this case. It would be generated on the fly in response to the query "Show all system attributes that satisfy the system cost requirements." What we see, then, is that every hardware and software component has an attribute that must contribute to the satisfaction of a system cost requirement. A third important category of traceability occurs with the linking of system requirements to domain requirements (i.e., a system cannot work until the system requirements have satisfied the relevant domain requirements). Figure 19 illustrates a scenario where requirements deal with the physical limitations of a network operating at 100 Mbps. In general, a certain type of network cable, CAT5, must be used in such a network. Further, this cable has a physical limitation of roughly 100 meters over which it can transport a signal. These domain requirements exist regardless of the system requirements. Since this system has a requirement to operate at 100 Mbps, the domain requirements become applicable, and must trace to system requirements. Figure 19 shows this trace, as well as the allocation of these system requirements to system component attributes. The relevant query might ask: "Show all relationships with system network requirements?" We see there is a connection from a network requirement (not specified by any domain requirements) to the Router component's WAN speed attribute.

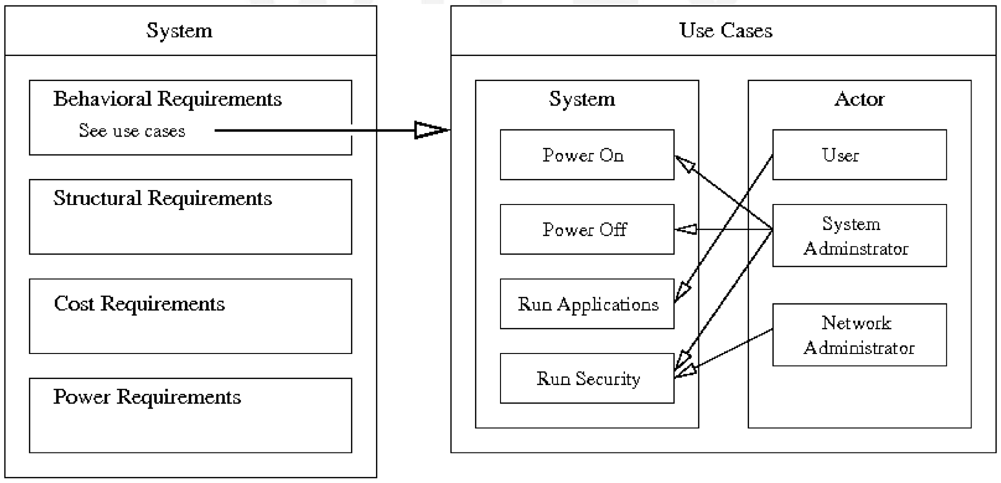


Figure 17. Office computing system higraph model: behavior requirements association with use cases.

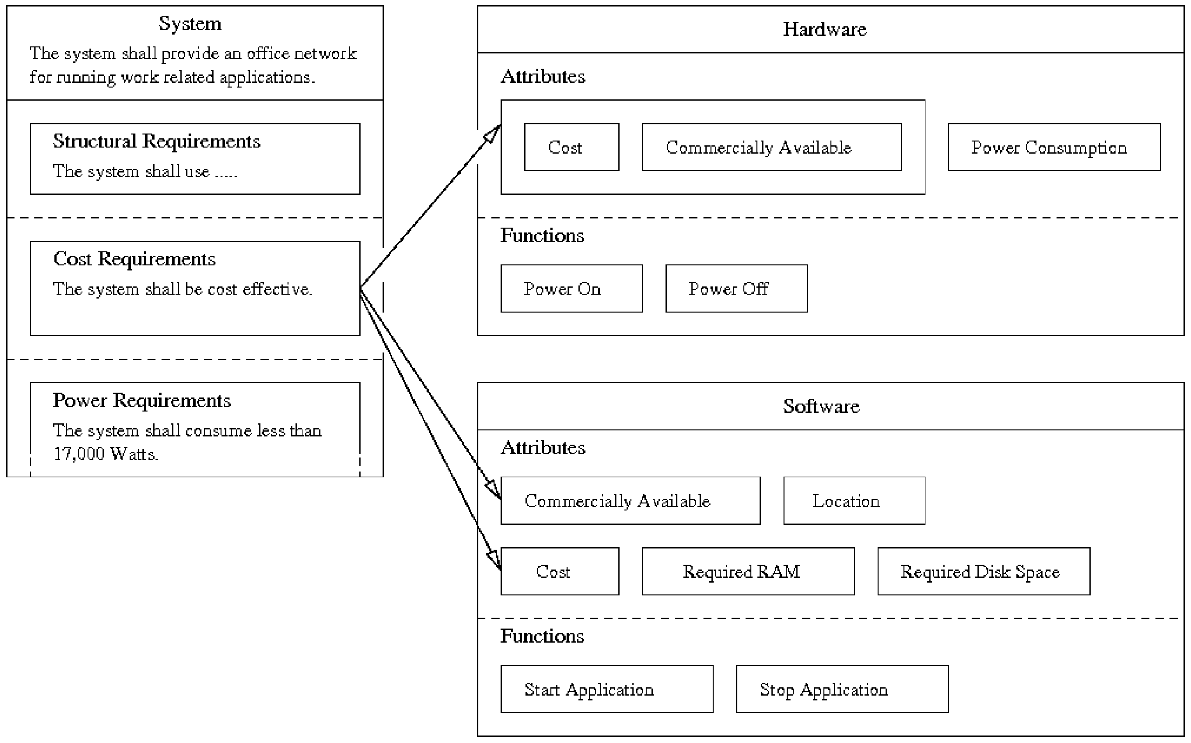


Figure 18. Office computing system: abbreviated higraph model of cost requirements allocation.

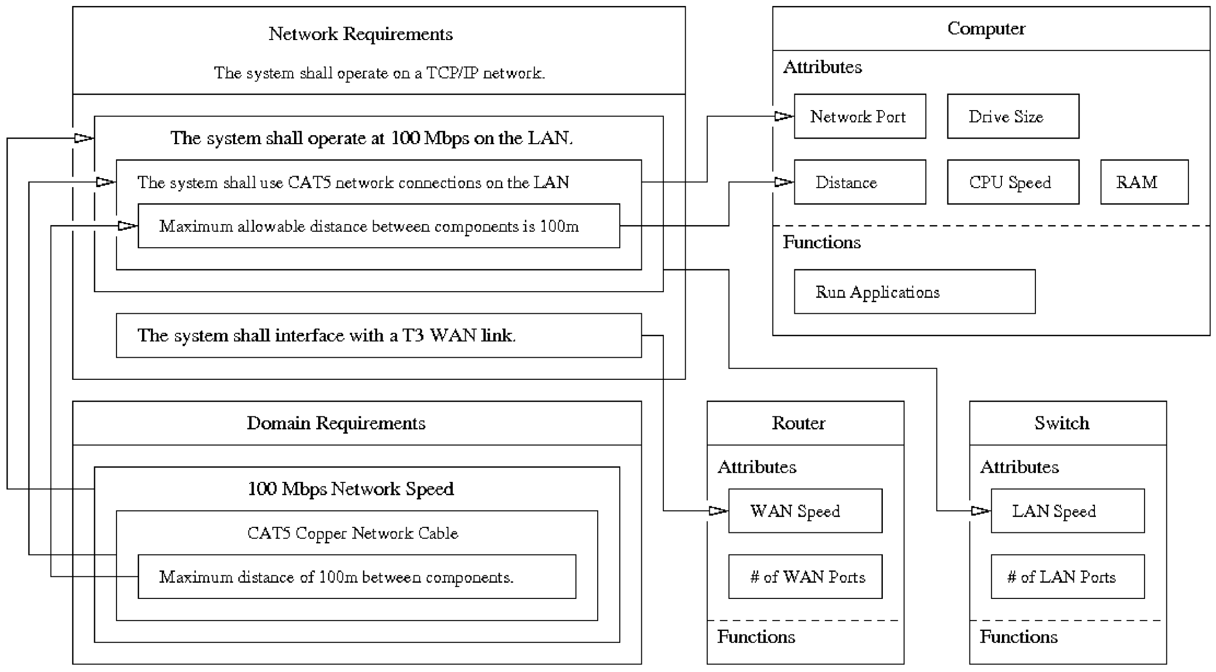

Network Requirements
The system shall operate on a TCP/IP network.
The system shall operate at 100 Mbps on the LAN.
The system shall use CAT5 network connections on the LAN
Maximum allowable distance between components is 100m
The system shall interface with a T3 WAN link.
Domain Requirements
100 Mbps Network Speed
CAT5 Copper Network Cable
Maximum distance of 100m between components.
Computer
Attributes
Network Port
Drive Size
Distance
CPU Speed
RAM
Functions
Run Applications
Router
Attributes
WAN Speed
# of WAN Ports
Functions
Switch
Attributes
LAN Speed
# of LAN Ports
Functions

Figure 19. Office computing system: domain requirements allocation.

6.2. Behavioral Requirements Traceability

Tracing behavior requirements to system states is only part of the design process. System behaviors that cause transitions into and out of system states have to be allocated to functions in system components. Like structure requirements and component attributes, behavior requirements trace through system behaviors to component functions, as illustrated by the comprehensive example shown in Figure 20. All of the functions that cause transitions into states in the "Email Running" behavior diagram must correspond to component functions in the system structure model. Functions are allocated to the e-mail software, POP3 software, and SMTP software components. It is important to note the direction of the colored edges. The edge comes from a system behavior (which causes a transition to a required system state) to a function in a system component. Also, the edges from the e-mail node to the POP3 and SMTP nodes imply inheritance (not allocation). This would be specified through a user's definition of edges used in the system higraph model. The Compose, Read, Receive, and Send e-mail behaviors are allocated to system component functions, but the StartApplication() behaviors remain unallocated. An appropriate query can establish these links, as shown in the upper half of Figure 20.

6.3. Mathematical and Logical Model

Sections 6.1 and 6.2 have focused on higraphs as a mechanism for visually conveying information. However, the real power of the higraph system model comes from the higraph quadruple $H = (B, E, \rho, \Pi)$. After the requirements, structure, and behavior models exist and are connected, a good way to begin construction of the math model is to define all of the possible meanings behind each node, edge, hierarchy, and orthogonal region. Tables III–VI show the nodes, edges, hierarchy classifications, and use of orthogonalization to represent the system model. Each table row defines a set corresponding to a particular logical organization. For instance, each node in any part of the Office Network higraph will fall into one of these sets. The set $B_{1\text{-}2\text{-}1\text{-}2}$ (system behavior requirements) consists of four nodes: Power On, Power Off, Run Applications, and Run Security (see Fig. 17). When the hierarchy portion of the model is defined (see details below), any nodes that fall under these four would also make up the set $B_{1\text{-}2\text{-}1\text{-}2}$.

Table IV shows a list of all of the logical definitions applied to edges. Again, each row in the table defines the logical sets of edges that make up the office network higraph model. For instance, the set E_6 (Satisfaction of a Domain Requirement by a System Requirement) consists of the three edges shown in Figure 19 that domain requirements to network requirements.

A set by itself is not terribly helpful. Even if all nodes are defined and grouped according to Table III, we still need to know where they fall in the higraph. These details are obtained from Tables V and VI, which specify the hierarchy and orthogonality organization of the higraph. As a case in point, ρ_3 (Association of Attributes with a Component) for the hardware component would be a set of three nodes (nodes of type Attribute—$B_{2\text{-}1\text{-}1}$): power consumption, cost, commercial availability. An example of a hierarchy set is Π_1 (requirements domain), which consists of four nodes: structural requirements, cost requirements, power requirements, and behavioral requirements (see Figs. 17 and 18). Of course, to know how anything relates to anything else in the system, we also need to know the set of edges.

6.4. Using the Office Network Higraph Model

A strength of the higraph model is the ability to query it to create custom views, elicit very specific information, or discover certain relationships among system requirements, behaviors, and components. These queries are really queries of the higraph quadruple stored within the higraph tables. Suppose, for example, that our requirement to interface the office network with a T3 WAN link (see Fig. 19) was changed to interface with a higher speed STM1 WAN link. What would the impact to the system be? We would query the model to find what relationships exist that can be traced to the requirement node $B_{1\text{-}1}$ (The system shall interface with a T3 WAN link). To do this, we would query the Edges set for any occurrence of $B_{1\text{-}1}$ (The system shall interface with a T3 WAN link). From our higraph equation, and from Figure 19 we determine that there exists an edge, $E_{5\text{-}1}$ [$B_{1\text{-}1}$ (The system shall interface with a T3 WAN link), $B_{2\text{-}1\text{-}1}$ (WAN Speed)]. As illustrated in Table III, these are attributes of nodes in the higraph structure. The query would then trace up through the hierarchy to find what component the $B_{2\text{-}1\text{-}1}$ (WAN Speed) attribute is allocated to.

How would the query know to perform this second trace to find an affected component? It's because the edge we found, $E_{5\text{-}1}$, has a meaning of "Assignment of a Structure Requirement to a Component Attribute." Moving up through the hierarchy from $B_{2\text{-}1\text{-}1}$ (WAN Speed) the query would find that $B_{2\text{-}1\text{-}1}$ (WAN Speed) belongs to the set ρ_3 (Router) = [$B_{2\text{-}1\text{-}1}$ (WAN Speed), $B_{2\text{-}1\text{-}1}$ (i.e., number of WAN Ports)]. These details can be found in Tables III and V. The relationship between WAN Speed and Router is illustrated in Figure 19. We now know that we have to modify the router component

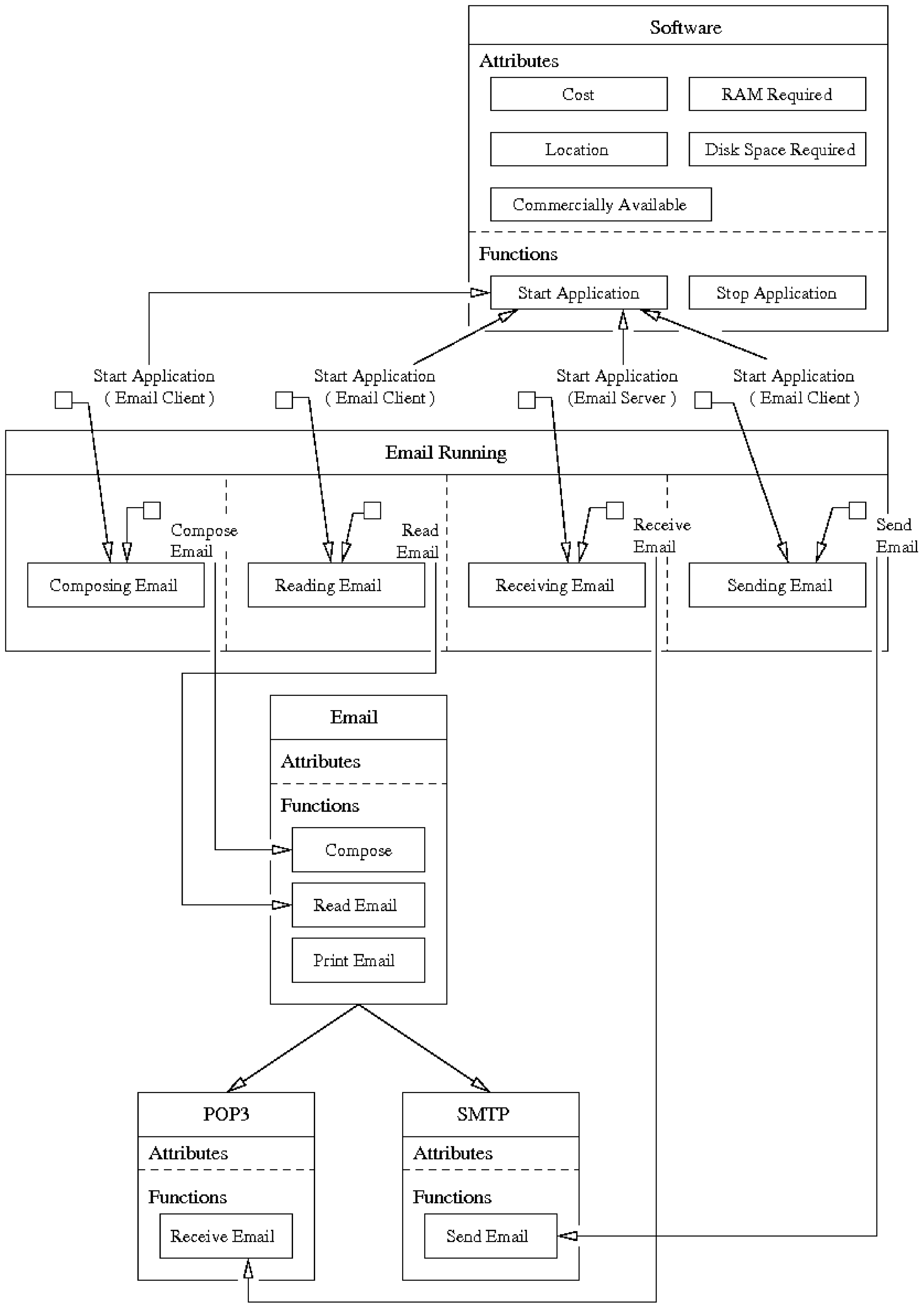


Figure 20. Office computing system: higraph model showing behavior allocation.

Table III. Office Network Higraph Model: Node Definitions

Area	Higraph Nodes (B)	Symbol
Requirements		
	Requirements Higraph	B_1
	Structure Requirements Higraph	B_{1-1}
	Requirement Number	B_{1-1-1}
	Requirement Area	B_{1-1-2}
	Requirement Type	B_{1-1-3}
	Requirement Text	B_{1-1-4}
	Requirement Owner	B_{1-1-5}
	Behavior Requirements Higraph	B_{1-2}
	Use Cases	B_{1-2-1}
	Actors	$B_{1-2-1-1}$
	System Behavior Requirements	$B_{1-2-1-2}$
Structure		
	Structure Higraph	B_2
	Components	B_{2-1}
	Attributes	B_{2-1-1}
	Functions	B_{2-1-2}
	Instances	B_{2-2}
Behavior		
	Behavior Higraph	B_3
	System States	B_{3-1}

to change the WAN speed to meet the new requirement. Once we modify/replace the Router component with one that meets this new STM1 WAN requirement, we would continue with trace that examines all edges coming from the router component to ensure no other requirements, structures, or behaviors have been adversely affected by our change. Such a trace would reveal, among other things, that we must remain within power and cost budgets (see Fig. 18). Does our new component satisfy these? The next trace to find all cost and power attributes from components within the system, sum them respectively, and evaluate those totals against the system requirements will provide us the answer.

Table IV. Office Network Higraph Model: Edge Definitions

Area	Higraph Edges (E)	Symbol
Requirements		
	Allocation of a User to a Behavior	E_1
Structure		
	Inheritance	E_2
	Multiplicity Association	E_3
Behavior		
	State Transition	E_4
System Level		
	Assignment	E_5
	Assignment of a Structure Requirement to a Component Attribute	E_{5-1}
	Assignment of a Behavior requirement to a Use Case	E_{5-2}
	Assignment of a Use Case to a System State	E_{5-3}
	Assignment of a State Transition to a Component Function	E_{5-4}
	Satisfaction of a Domain Requirement by a System Requirement	E_6

Table V. Office Network Higraph Model: Hierarchy Definitions

Area	Higraph Hierarchy (ρ)	Symbol
Requirements		
	Requirements Hierarchy	ρ_1
	Use Case Hierarchy	ρ_2
Structure		
	Association of Attributes with a Component	ρ_3
	Association of Functions with a Component	ρ_4
Behavior		
	Behavior Hierarchy (States/Substates)	ρ_5

There are, of course, an almost infinite number of possibilities for queries against the system higraph model. In an industrial setting, many queries would result from changed requirements, but others may result from stakeholder information requests (e.g., finance wants to know what the total cost of the system is) or equipment obsolescence (e.g., a certain software package has reached its end of life). Once implemented in software, the series of traces and evaluations to provide the results of a query will be as automated as possible based on the user defined tables for nodes, edges, hierarchy, and orthogonality, and changes users make to the. In this manner, the higraph model serves not only to present information, but to show and validate how the system is put together.

7. CONCLUSIONS AND FUTURE WORK

Higraphs are a useful tool for organizing and connecting data and information generated during the system engineering lifecycle. They can be defined mathematically and logically, which clears any ambiguities from the system model, as well as allows for the system model to be "smart" in the way it responds to queries for specific information. The data that are presented as a result of a query on the system model can be used by system engineers to make knowledgeable design, implementation, operational, and support decisions for the system. Unfortunately, these benefits do not come without costs. Because components from anywhere in a system model can have a relationship (connection) to components anywhere else in that system model, higraph models can quickly become very detailed, presenting engineers with too much data and information to work with simultaneously at any one time. For a given higraph, the process of arranging the nodes and edges in a visual layout that maximizes communication of information to an end-user is far from trivial. Harel says of this issue [Grossman and Harel, 1997]: "In practice, overlaps should probably be used somewhat sparingly, as overly overlapping blobs might detract from the clarity of the total diagram." Still this solves only part of the problem. To mitigate the possibility of overwhelming the end user, there must be a filter (or abstraction tool) that mines the higraph and presents only the desired information to the end-user in response to specific queries. In other words, in order for such a methodology and tool to be useful in industry, higraph modeling languages must be able to interface with a software tool to perform this filtering.

Looking forward, any software tool that implements higraphs would, at a minimum, have to allow the following tasks: (1) create a system requirements higraph

Table VI. Office Network Higraph Model: Orthogonality Definitions

Area	Higraph Orthogonality (Π)	Symbol
Requirements		
	Requirements Domain	Π_1
Structure		
	Hardware Component or Software Component	Π_2
	Component Attribute or Component Function	Π_3
Behavior		
	Allowed Concurrent Behavior	Π_4

from user inputs, (2) create a system structure higraph from user inputs, (3) create a system behavior higraph from user inputs, (4) allow the user to define types of nodes, edges, hierarchies, and orthogonalities, and (5) allow the user to connect nodes via edges. Generating these viewpoints is a matter of following a select group of edges from specific nodes in the higraph. Suppose, for example, that an engineer needs to find all requirements associated with a specific system component, He/she only needs to trace all "requirements" emanating coming from the component node in the higraph. Likewise, the costs associated with a specific subsystem can be retrieved by pulling all of the cost attributes from the components that make up this subsystem. The interfaces available to create and define these things could vary. User inputs could come from XML forms, spreadsheets, text files, or developed graphical user interfaces (GUI's). Translation rules could be applied to import existing artifacts (e.g., class diagrams, statecharts, etc.) into a new higraph model. Transformation tools like XSLT [XSLT, 2002] could conceivably be used to automatically generate UML/SysML diagrams—thus, a software environment where higraphs and UML/SysML representations coexist certainly seems feasible. To the extent possible, another software tool could automate the translation of UML/SysML diagrams to higraphs, and vice versa.

REFERENCES

M. Austin, Lecture Notes for ENSE 621/ENPM 641: Visual modeling of engineering systems with UML, September 2002.

M.A. Austin, V. Mayank, N. Shmunis, and R.M. Paladin, Graph-based visualization of requirements organized for team-based design, Syst Eng 9(2) (2006), 129–145.

A. Bell, Death by UML fever, ACM Queue 2(1) (2004), http://www.acmqueue.com.

K. Berkenkotter, Using UML 2.0 in real-time development: A critical review, 2003. See http://www-verimag.imag.fr.itemBlackP.,DirectedAcyclicGraphs–from DictionaryofAlgorithms and Data Structures, 2006. See http://www.nist.gov/dads/HTML/directAcycGraph.html.

V. Chachra, P.M. Ghare, and J.M. Moore, Applications of graph theory applications, Elsevier North Holland, New York, 1979.

A. Dupagne and A. Teller, Hypergraph formalism for urban form specification, COST C4 Final Conf, Kiruna, September 21–22, 1998.

K. Fogarty, System modeling and traceability applications of the higraph formalism, MS Thesis, Institute for Systems Research, University of Maryland, College Park, May 2006.zaq;3

K. Fogarty and M. Austin, System modeling and traceability applications of the higraph formalism, ISR Technical Report XXX, Institute for Systems Research, University of Maryland, College Park, August 2007, p. 49.

O. Grossman and D. Harel, On the algorithmics of higraphs, Technical Report, Rehovot, Israel, 1997.

D. Harel, Statecharts: A visual formalism for complex systems, Sci Comp Program 8 (1987), 231–274.

D. Harel, On visual formalisms, Commun ACM 31 (1988), 514–530.

Headway Software Inc., Closed Loop Development with Headway ReView,zaq;4 June 2001.

P. Letelier, A framework for requirements traceability in UML projects, 1st Int Workshop Traceability Emerging Forms Software Eng, in conjunction with 17th IEEE Int Conf Automated Software Eng, 2002.

Microsoft Corporation, Microsoft Office Visio Standard CDROM,zaq;4 2003.

M. Minas, Hypergraphs and a uniform diagram representation model, Proc 6th Int Workshop Theory Appl Graph Transformations (TAGT, 98), Paderborn, Germany, 1998.

D. Muller, Requirements engineering knowledge management based on STEP AP233, 2003.zaq;5

T. Munzner, Interactive visualization of large graphs and networks, Ph.D. Thesis, Stanford University, Palo Alto, CA, 2000.

D. Oliver, AP233—INCOSE status report, INCOSE INSIGHT 5(3) (October 2002).

The Object Management Group, What is OMG-UML and why is it important, http://www.omg.org/news/pr97/umlprimer.html, 1997.zaq;3

R.F. Paige, Heterogeneous specifications and their application to software development, Technical Report, Toronto, Canada, August 1995.zaq;6

M. Ramaswamy and S. Sarkar, Using Directed hypergraphs to verify rule-based expert systems, IEEE Trans Knowledge Data Eng 9(2) (1997), 221–237.

Rational Software Corporation, Microsoft Software Corporation, UML summary, Version 1.1, http://umlcenter.visual-paradigm.com/umlresources/summ 11.pdf, September 1997.

SysML Partners, Systems Modeling Language specification, Version 0.9 DRAFT, http://www.sysml.org/artifacts/specs/SysML-v0.9-PDF-050110.zip, January 2005a.

SysML Partners, Systems Modeling Language specification, Version 1.0 alpha, http://www.sysml.org/artifacts/specs/SysMLv1.0a-051114R1.pdf, November 2005b.

Teamcenter (formallySLATE), http://www.plm.automation.siemens.com/en us/products/teamcenter/, 2008.

T e l e l o g i c , A B . T e l e l o g i c D O O R S , http://www.telelogic.com/corp/products/doors/doors/index.cfm, 2006.

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

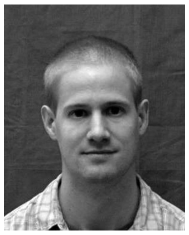
UML Forum, UML FAQ, http://www.uml-forum.com/faq.htm, 2006.

UML Glossary, Computer Science Department, California State University, San Bernardino, CA, http://www.csci.csusb.edu/dick/samples/uml.glossary.html, June 2006.

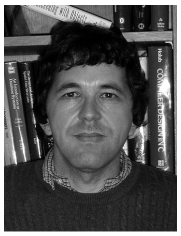
Wiki, Graph theory at Wikipedia, http://en.wikipedia.org/wiki/Graph theory, 2006.

M. Wissen and J. Ziegler, A methodology for the component-based development of Web applications, Proc 10th Int Conf Human-Computer Interaction (HCI International 2003), volume 1, Crete, Greece, 2003.

XML Stylesheet Transformation Language(XSLT), http://www.w3.org/Style/XSL, 2002.



Kevin Fogarty is a Senior Systems Engineer with Science Applications International Corporation (SAIC) working out of their Columbia, MD office. He specializes in systems integration, with a focus on optical communications technology. Mr. Fogarty holds a Masters Degree in Systems Engineering from the University of Maryland (College Park) and a Bachelors degree in Computer Engineering from Virginia Tech.



Mark Austin is an Associate Professor at the University of Maryland, College Park, with joint appointments in the Department of Civil and Environmental Engineering and the Institute for Systems Research (ISR). Mark is past director of the Program for Master of Science in Systems Engineering (MSSE) at ISR. He is also Associate Director of the Systems Engineering and Integration Laboratory at ISR, and past co-chair of ICOSE's Commercial Practices Interest Working Group (CPIWG). His Ph.D. and M.S. degrees are from the University of California Berkeley. He also has a B.E. (First Class Honors) in Civil Engineering from the University of Canterbury, New Zealand.

<enote>AQ1: Please supply Key words
<enote>AQ2: Please list in References
<enote>AQ3: Please cite in text
<enote>AQ4: Please give city
<enote>AQ5: More information needed: Report? Journal article?
<enote>AQ6: Please give sponsoring institution