System Modeling and Traceability Applications of the Higraph Formalism

Article *in* Systems Engineering · March 2009

DOI: 10.1002/sys.20113

2 authors:

Kevin Fogarty
1 PUBLICATION **9** CITATIONS
SEE PROFILE

Mark Austin
University of Maryland, College Park
100 PUBLICATIONS **973** CITATIONS
SEE PROFILE

ABSTRACT

Title of Document: SYSTEM MODELING AND TRACEABILITY APPLICATIONS OF THE HIGRAPH FORMALISM

Kevin Fogarty, Master of Science in Systems Engineering, 2006

Directed By: Dr. Mark Austin, Department of Civil and Environmental Engineering, and The Institute for Systems Research, University of Maryland at College Park

One of the most important tools for a systems engineer is their system model. From this model, engineering decisions can be made without costly integration, fabrication, or installations. Existing system modeling languages used to create the system model are detailed and comprehensive, but lack a true ability to unify the system model by showing all relationships among all components in the model. Higraphs, a type of mathematical graph, allow systems engineers to not only represent all required information in a system model, but to formally show all relationships in the model through hierarchies, edges, and orthogonalities. With a higraph system model, all relationships between system requirements, components, and behaviors are formalized allowing for a “smart” model that can be queried for custom sets of information that, when presented to the systems engineer, will aid in engineering decisions.

SYSTEM MODELING AND TRACEABILITY APPLICATIONS OF THE HIGRAPH FORMALISM

By

Kevin Fogarty

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science in Systems
Engineering
2006

Advisory Committee:
Dr. Mark Austin, Chair
Dr. John Baras, Committee Member
Dr. David Lovell, Committee Member

Dedication

This paper is dedicated to Brayden Thomas. May you grow up to be strong and faithful, and may you get to experience and learn all this world has to offer.

Acknowledgements

I would like to acknowledge the following people, all of whom contributed to this paper.

Dr. Mark Austin, University of Maryland at College Park.

- Associate Professor, Department of Civil and Environmental Engineering, and
- Faculty, The Institute for Systems Research

Dr. Austin, in his role as the thesis advisor, has provided overall guidance, editorial comment, and technical input to this thesis. Dr. Austin served as the Chair of this Thesis Committee.

Dr. John Baras, University of Maryland at College Park.

- Professor, Department of Electrical and Computer Engineering,
- Founding Director, The Institute for Systems Research, and
- Director, Maryland Hybrid Networks Center

Dr. Baras served as a member of this Thesis Committee, and has provided technical input to this thesis.

Dr. David Lovell, University of Maryland at College Park.

- Associate Professor, Department of Civil and Environmental Engineering, and

- Faculty, The Institute for Systems Research

Dr. Lovell served as a member of this Thesis Committee, and has provided technical input to this thesis.

Chad Rivera, University of Maryland at College Park.

- Past student, Master's of Engineering, Systems Engineering Program

Mr. Rivera worked with Mr. Fogarty on a paper that served as the initial effort to research the higraph formalism as a tool for system modeling and traceability.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Section 1.1: Introduction and Objectives

It is well known that system modeling is a fundamental step in the Systems Engineering process. Modeling techniques allow for the comprehensive representation, organization, design, and evaluation of a system, from 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."

And how do we create this blueprint? We create it through the use of system modeling languages. Unfortunately, this is where the grand vision of system modeling and the reality of 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 drive methods) are relatively complex, and are difficult to use beyond the system modeling phase of the systems engineering lifecycle. In commercial/industrial settings, modeling languages (in the form of popular commercial tools) are often forced into use by management on engineering projects. Reference [4] provides a satirical, but all too true, account of this trend. 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 semi-automation) 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. The result will be separate, disjoint models for all of these things; in other words, there is never a complete unified system model.

A second problem is that models created with existing languages and tools often serve limited functionality during the implementation, operation, and maintenance phases. Present day tools such as DOORS for requirements management [26], or Visio for UML model management [14], allow for manual updates to models as things change during the system's lifecycle, but what is really needed is a model that can automatically be manipulated to show the propagation of cause-and-effect relationships and sensitivity of effects to perturbations in causes (e.g., adjustments to requirements, components, and/or behaviors).

The goals of this thesis are to mitigate these limitations by exploring the feasibility and potential benefits of a new system modeling language, capable of representing a complete (structure, behavior, requirements) system model, plus levels of automatic manipulation to aid in change management. The foundation for this work is higraphs, a topo-visual modeling formalism introduced by David Harel in 1988 [9]. To date, the higraph formalism has been applied to a wide range of applications including:

1. Statecharts in UML [10],
2. Expression of relationships in drawings [25] and urban forms [7],
3. Formal specifications in software development [21],
4. Component-based development of web applications [30], and
5. Verification procedures in rule-based expert systems [22]

For systems engineering, we hope that this new system modeling language will provide a unified system model in the sense of supporting systems engineering activities through the entire system lifecycle.

Section 1.2: Organization of Thesis

This thesis begins with a detailed introduction to our current understandings of higraphs: What are they, what are they currently used for, and what formal definitions (mathematical models) exist for Higraphs?

Chapter 3 focuses on existing system modeling methods, specifically the Unified Modeling Language (UML) [11, 12] and the Systems Modeling Language (SysML) [15, 16]. To put the role of visual modeling languages in systems engineering in context, a brief background on UML is presented, along with summaries of its capabilities and comments on its weaknesses. SysML is discussed in a similar fashion. The most important is comments on how a higraph implementation of a system model would improve on specific weaknesses identified in the analysis of SysML and UML.

Chapter 4 focuses on using Higraphs as a complete system modeling tool. We will show how Higraphs may be used to model system requirements, system structure, and system behavior; how these models incorporate hierarchy and orthogonality; and how each model can be connected to the others in a useful (and formal) manner. Deficiencies of the higraph model are also covered in this chapter.

Chapter 5 extends the basic mathematical definition of Higraphs and discusses how this definition can be modeled and used by computer software. We will then discuss some of the most beneficial applications that could be derived from a complete, formally defined, higraph model.

Chapter 6 discusses practical applications of higraph models. It answers the question, once the higraph model is created, what can you do with it? We will show that the higraph model acts as a "smart" model in that it can respond to queries, and present engineers with customized information about the system.

Chapter 7 documents an example of creating a Higraph model of an (existing) system, and demonstrates smart queries of the system model.

Chapter 8 covers conclusions and future work, including work that is needed to further advance higraphs as a system modeling tool.

Section 1.3: Contributions

The contributions of the work described in this thesis include:

- A thorough examination of the higraph formalism and how it can be applied to system modeling efforts,
- An extension to the current mathematical and logical definition of a higraph in order for it to be applicable to systems engineering efforts, and
- A description of how a higraph model of a system can be used to respond to user queries, and present customized "viewpoints" of the system.

Chapter 2: Introduction to Higraphs

This chapter introduces the mathematical formalism (structure and properties) for higraphs. It also describes current applications of higraphs, and commercial software enabled by higraphs.

Section 2.1: Higraph Definition

A Higraph can be most easily defined as a mathematical graph that combines depth and orthogonality. In other words:

$$\text{Higraph} = \text{Graph} + \text{Depth} + \text{Orthogonality}\ [8]$$

Equation 1: Definition of a Higraph

To start with the basic underlying concepts, an informal definition of a mathematical graph is "a set of objects called vertices (or nodes) connected by links called edges (or arcs) which can be directed (assigned a direction)." [29]

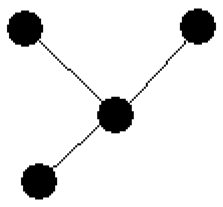


Figure 1: Example Mathematical Graph

Figure 1 shows, for example, a small mathematical graph. The graph 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. As we will soon see in this paper, applying meaning to the equivalent nodes and edges of a Higraph can serve a useful purpose in systems engineering.

The term Depth in Equation 1 can be thought of as a defined hierarchy, and the term Orthogonality can be thought of as a Cartesian product [9] or partitioning. While we can show that Equation 1 is complete in defining the mathematical definition of a higraph, we must also incorporate the notion of Euler Circles (or Venn Diagrams) to define the "enclosure, intersection, and exclusion" [9] of elements in a Higraph.

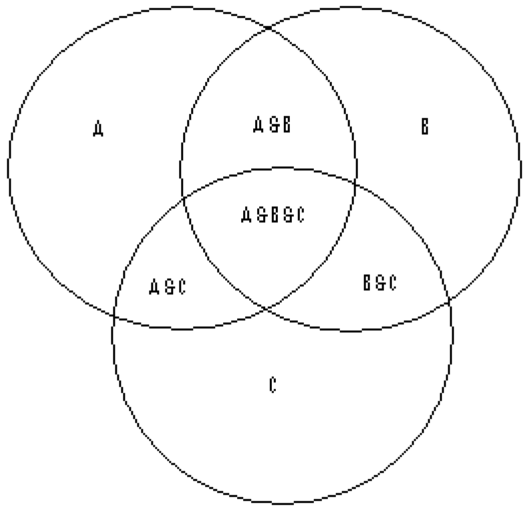


Figure 2: Example Venn Diagram

The Venn diagram in Figure 2 shows the relationships between set A, set B, and set C. These sets are defined by the enclosures around them. Where set A and set B intersect, we see "A & B", and this implies the exclusion of set C from this space. This notion of enclosure, inclusion, and exclusion defines the relationships between

A, B, and C in a given diagram. Likewise, the notion of enclosure of one set within another creates an implied hierarchy. But for higraphs, enclosure more than implies a hierarchy, it defines one.

Enclosure and Hierarchy in Higraphs

In a higraph, we combine the notion of a graph node (Figure 1), and an enclosure in a Venn diagram (Figure 2), to form the atomic (lowest level) element in a higraph. Harel refers to this atomic element as a *blob* [9]. In Figure 3 below, a graph structure is defined through connectivity relationships among the four blobs. In this case, each blob has some sort of relationship (connectivity) to the central blob, Blob A. Figure 3 actually represents the same information as Figure 1 (Basic Mathematical Graph), but uses Harel's blobs, instead of traditional nodes.

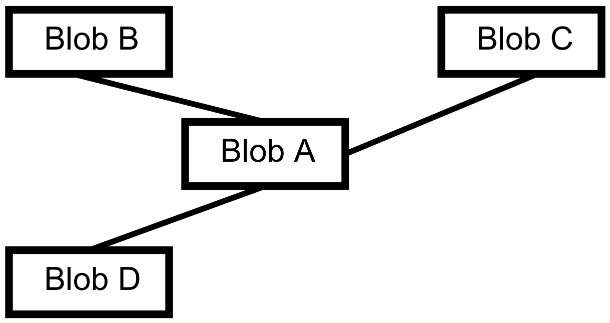


Figure 3: Graph showing Atomic Elements

By combining the Venn diagram notion of enclosure and the higraph notion of depth, we can develop and document hierarchies. One *blob* within another *blob* shows a hierarchical relationship, and could be equivocated to a directed edge in a graph.

A Directed Graph is a graph in which each graph edge is assigned a direction. Figure 4 shows a Directed Graph where a hierarchy among the nodes is defined by the directed edges connecting them. When we swap nodes for blobs, and add areas of enclosure, we get a "Directed" Higraph. The "Directed" Higraph shows the same hierarchy between the blobs (nodes) as the Directed Graph. To arrive at a Higraph, we need only remove the directed edges between the blobs from the "Directed" Higraph. By its definition, the depth, or hierarchy, shown by a Higraph is defined by the enclosure of one blob within another. The Higraph in Figure 4 shows the same information as the Directed Graph in Figure 4.

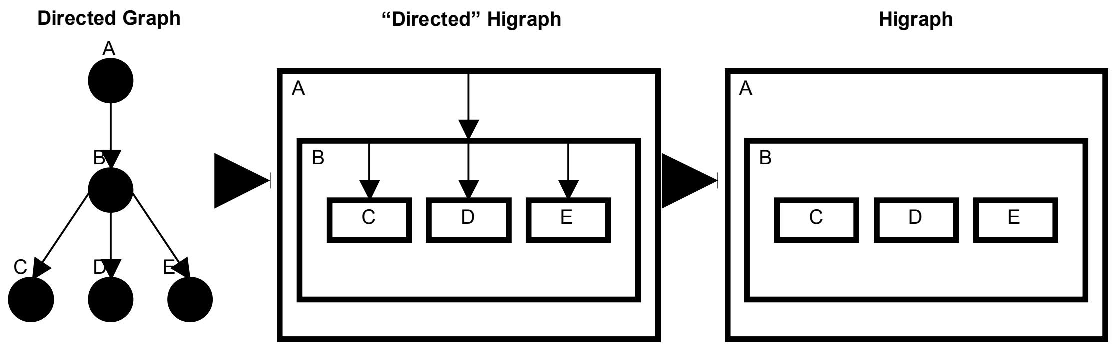


Figure 4: Developing a Higraph from a Directed Graph

(For purposes of clarity—and aesthetics—we will refer to Higraph blobs simply as nodes in the remainder of this paper.)

Edges in Higraphs

From the original definition of a higraph as a graph (reference Equation 1.1), edges can connect any node to any other node in order to represent various relationships (physical connections, logical connections, etc.). Figure 5 below shows a generic example of allowed connectivity in a Higraph. This connectivity can exist in any user defined way—from a node in any level of the hierarchy, to node in any other.

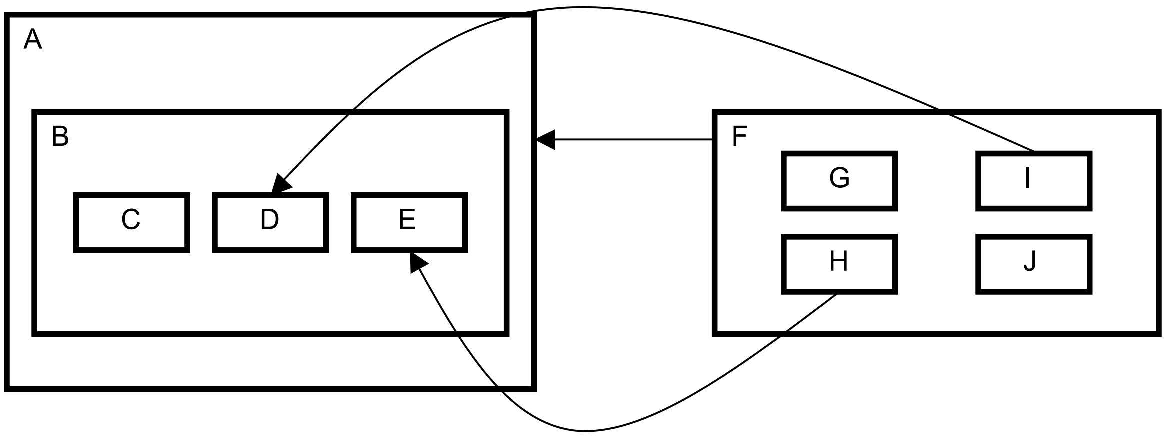


Figure 5: Higraph with Edges

Notice that nodes B and F contain the set of nodes {C, D, E} and {G, H, I, J}, respectively. The organization of nodes illustrates the hierarchical organization of data/information in higraphs.

Orthogonality in Higraphs

The notion of orthogonality—Cartesian product as defined by Harel—is the final element in the higraph equation. For our purposes, orthogonality will be shown as a distinct partition, or dashed line, within a higraph. Figure 6 below shows an example of orthogonal nodes.

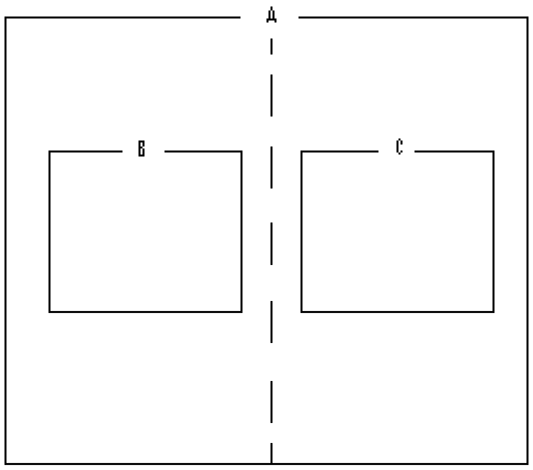


Figure 6: Higraph with Orthogonal Components

Systems Engineering Example of a Higraph

As previously mentioned here, applying a specific meaning to nodes, edges, and now the orthogonal regions in a higraph, can have applications in systems engineering. Figure 7 below shows a higraph that includes hierarchy and orthogonality. In the commonly found team-based development of a system used in industry, requirements will be organized into groups that can be assigned to teams having expertise in an area relevant to those requirements. Some of these requirements will be satisfied in an autonomous manner (i.e., the team will find a solution without having to consult with other teams). Other requirements will correspond to areas of concern that apply to multiple disciplines and/or lie at the interfaces between sub-systems. As a case in point, Figure 7 shows an example of how the higraph formalism can be used to group requirements in a logical manner using hierarchy and orthogonality.

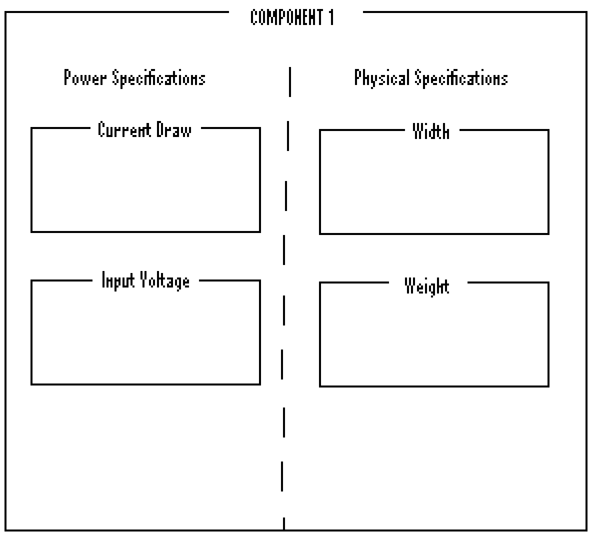


Figure 7: Defining Higraph Nodes and Regions for a Systems Engineering Application

The labels on the nodes in this Higraph (COMPONENT1, Current Draw, Input Voltage, Width, and Weight) provide semantic meaning top the hierarchical organization of nodes defined in Figure 7. The current draw, input voltage, width, and weight are all "lower level" specifications of the "higher level" COMPONENT1. Two orthogonal regions, Power Specifications and Physical Specifications, are defined within COMPONENT1. Each region contains two nodes, and again the location in of each node in one orthogonal region or the other has meaning—current draw and input voltage are power specifications of COMPONENT1, and width and weight are physical specifications of COMPONENT1.

Topology in Higraphs

One last important point to note, per Harel's definition in [9], is that higraphs are topo-visual formalisms. As expressed in Munzner [17], the importance of a topo-visual formalism is found in "non-metric topological connectedness as opposed to pure geometric distance." In other words, for higraphs, what is important is the topology between the nodes—how they are arranged—and the connectivity between the nodes—how the edges are connected. The size and physical distance between nodes in a Higraph is unimportant. To illustrate this point, Figure 8 below replicates the data/information presented in Figure 7 above. Again, what is important are the topological—or spatial and connectivity—properties of the Higraph. These topological properties are the depth (hierarchy represented by enclosures), orthogonality, and connectivity shown in a higraph.

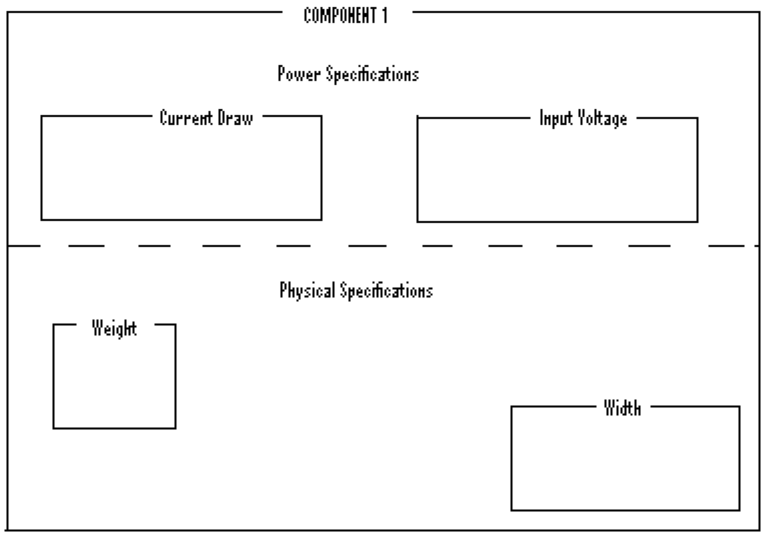


Figure 8: Different Visual Representation of Figure 7

Section 2.2: Existing Mathematical and Logical Models of Higraphs

As a follow up to the "On Visual Formalisms" paper [9], Harel provides a comprehensive model for defining the mathematical properties of higraphs [8]. These properties will be extremely instrumental in our proposition that higraphs can serve as a useful tool for complex system modeling by creating a unique mathematical definition of a higraph. In turn, this mathematical definition can be ingested and used by proposed higraph modeling software tools to work with the higraph model. For now, and as a starting point, we will simply present what Harel has provided.

Existing Mathematical Definition of a Higraph

The basic mathematical definition of a higraph can be summarized as follows [8]:

- 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

In [8], Harel provides the lowest level definitions of the hierarchy and partitioning functions.

Example Mathematical Definition of a Higraph

Applying these definitions to the higraph shown in Figure 9 yields the following equations:

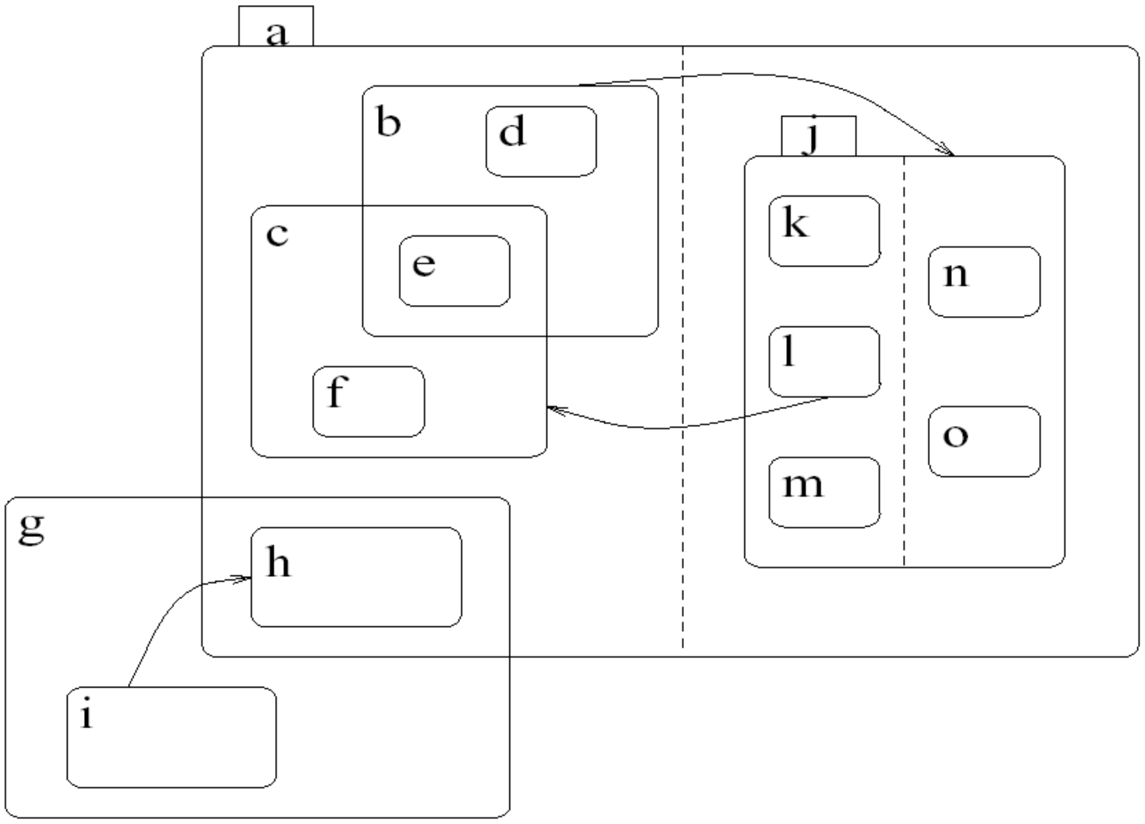


Figure 9: Example Higraph for Math Modeling

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. ρ(H) = Σ ρ(b ε B)
    a. ρ(a) = {b, c, h, j}
    b. ρ(b) = {d, e}
    c. ρ(c) = {e, f}
    d. ρ(g) = {h, i}
    e. ρ(j) = {k, l, m, n, o}
    f. ρ(d) = ρ(e) = ρ(f) = ρ(h) = ρ(h) = ρ(k) = ρ(l) = ρ(m) = ρ(n) = ρ(o) = 0

4. $\Pi(H) = \Sigma\, \pi_n(b\ \varepsilon\ B)$
    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. $\pi_1(b) = \pi_1(c) = \pi_1(d) = \pi_1(e) = \pi_1(f) = \pi_1(g) = \pi_1(h) = \pi_1(i) = \pi_1(k) = \pi_1(l) = \pi_1(m) = \pi_1(n) = \pi_1(o) = 0$

This set of equations completely defines the mathematical model defining higraph H, shown in Figure 9.

Directed Acyclic Graphs and Higraphs

From any Higraph, or set of equations that define a Higraphs, you can derive a Directed Acyclic Graph, or DAG. An example DAG is shown in Figure 10 below. The basic definition of a DAG as defined by the National Institutes of Standards and Technology [6] is: "A directed graph with no path that starts and ends at the same vertex."

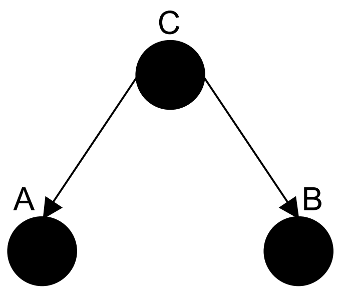


Figure 10: Sample Directed Acyclic Graph (DAG)

The DAG representation is useful because it can be derived from the higraph quadruple using algorithms, which in turn lends itself to computer/software modeling of higraphs. From a DAG model, we can do two very helpful things:

1. Trace through the DAG based on unique rules (or search criteria)
2. Construct a higraph

In [8], a configuration is defined as the set of nodes corresponding to the vertices constituting a legal trace of the [DAG] higraph. The legal trace will be the result of some rule or command that causes the trace.

Example DAG Representation of a Higraph

For instance, from Figure 9, there are two valid traces that will return nodes n and o: (A→J→N) and (A→J→O). The command that executes this trace would be to find all components in the "2nd" 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.

By employing the concept of XOR decomposition, the second useful purpose that the DAG model serves is the ability to construct a traditional higraph [8]. Harel defines the concept of XOR as it relates to DAGs as:

"If a and b are non-intersecting and are contained in c, and c contains no other blobs, then c is the XOR of a and b; and dually, if a and b intersect and their intersection contains blob a and none other, the c is also the XOR of a and b."

The appeal of XOR decomposition to systems engineering is that they allow for top-down and bottom-up representation of systems organization. The former case is illustrated below in Figure 11. While Figure 12 shows the application of a bottom-up decomposition.

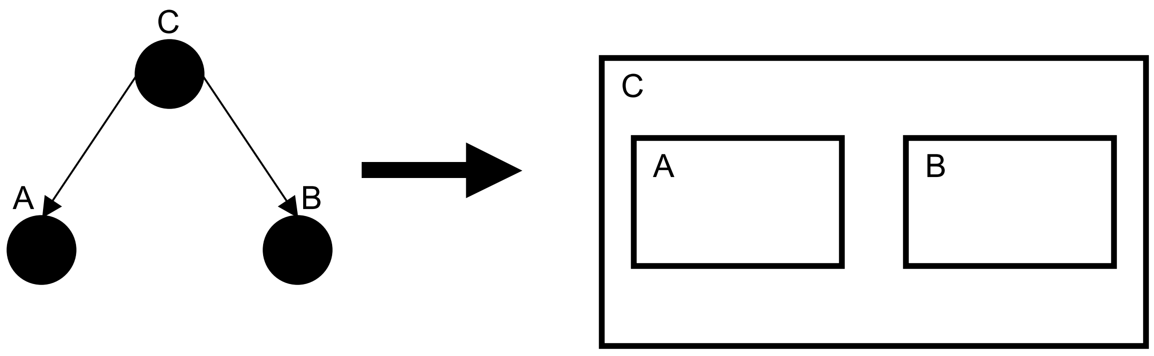


Figure 11: XOR Decomposition of DAG: Top-Down

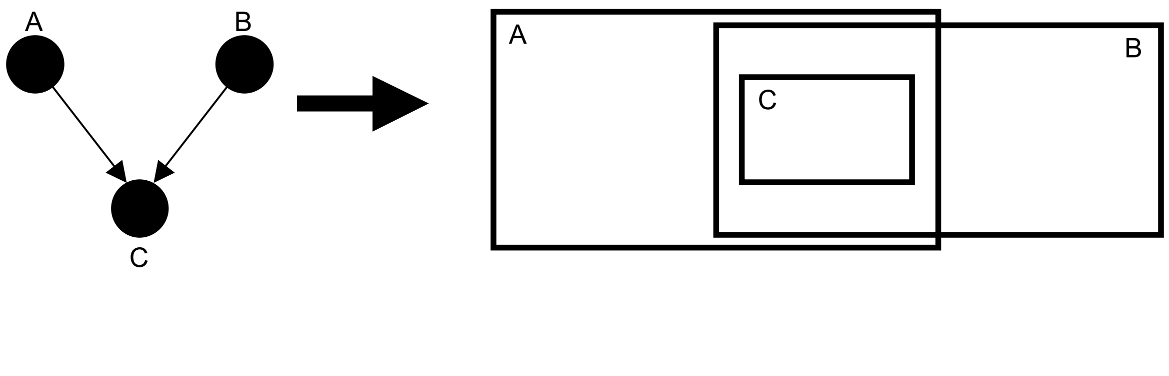


Figure 12: XOR Decomposition of DAG: Bottom-Up

Summary of the Mathematical and Logical Definition of a Higraph

Together the Higraph quadruple (B, E, ρ, Π) and the DAG model mathematically and logically define a higraph. Later in this paper, we will extend these mathematical and logical concepts in order to apply them to Higraphs used specifically for complete system modeling. First, however, we will examine the existing uses of Higraphs.

Section 2.3: Existing Uses of Higraphs

While most of the discussion and research around higraphs has been primarily theoretical to this point, there are a few practical uses of higraphs. See [9] and [11] for examples of this.

Higraphs used in/as Statecharts

We see higraph like characteristics in Statecharts and UML State Diagrams. These commonly accepted system models are often used in real-world engineering projects to define system behaviors. As a matter of fact, statecharts really are the basis for Higraphs [9]. The characteristics of higraphs are evident in traditional Statechart modeling, as well as in the state diagrams of UML. Nodes represent allowed system states, and edges represent transitions between states (system functions). Hierarchies can be shown through enclosure, and concurrency is modeled with orthogonality.

Higraphs used in Commercial Software

Higraphs have also made their way into commercial software tools. An example of a commercial tool implementing higraph concepts and characteristics is Headway

Software's reView [11]. Headway Software, an Ireland based software company, implements higraph like modeling in its software product Headway reView. Their white paper, Closed Loop Development [11], discusses one approach to enterprise level software development in which higraphs are implemented to aid in the development process. They contend that in dealing with a large code-base (the source code, libraries, packages, etc. that make up a commercial software product) there are four criteria that a tool would have to meet to maximize efficiencies:

1. Layering: "All the elemental dependencies are rolled up to provide an accurate view of the dependency structure at any level (layer) in the code-base."
2. UML-Lite: "Meaningfully present all dependencies at any level."
3. Layouts: Visually "present relational information."
4. Multiple Views: "Provide a number of parallel and linked views…each of which shows different information relative to the user's current focus."

To meet these for criteria, Headway implements higraphs "at the core of its solution." Although thorough testing of this product is not feasible under the guise of an academic thesis, there are some big name companies (Sun Microsystems, TRW, Delta Airlines) who have had positive things to say about this software, and the way this software, using the higraph model, allows engineers/programmers to understand and visual the layers and relationships in a complex software system [12].

Potential for Higraphs to be Used as a System Modeling Tool

Based on the few existing uses of higraphs, theories presented in earlier papers, and our efforts to evaluate the higraph formalism, we contend that higraphs have the potential of being a useful tool for complete system modeling. By "complete system modeling" we mean that the underlying formalism and supporting software tools should be able to create and display general connectivity and organizational (hierarchical and orthogonal) relationships among system requirements, and elements of system structure and system behavior models. To put the higraph formalism in the context of a systems engineering domain, values (numerical and textual) would be assigned to the higraph nodes (in domain areas such as hardware and software, electrical and mechanical, all four, or many more) and edges. Indeed, we speculate for design of multidisciplinary systems, higraphs might be a particularly strong choice for representing dependencies and relationships among multiple aspects of the model.

Chapter 3: Existing System Modeling Languages

This chapter evaluates the strengths and weaknesses of the Unified Modeling Language and the Systems Modeling Language, two current systems modeling languages used by systems engineers for visual modeling of systems. From this perspective, we will compare and contrast Higraphs as a similar tool.

Section 3.1: The Unified Modeling Language

Subsection 3.1.1: What is The Unified Modeling Language?

The development of Unified Modeling Language (UML) began in 1994 through efforts at Rational Software Corporation to unify the Booch and OMT (Object Modeling Technique) methods. The major goals of this effort were:

- Enable the modeling of systems (and not just software) using object-oriented concepts.
- Establish an explicit coupling to conceptual as well as executable artifacts.
- Address the issues of scale inherent in complex, mission-critical systems.
- Create a modeling language usable by both humans and machines. [20]

Prior to the existence of UML, a number of visual modeling languages were in use by industry; this lead to incompatibility and inefficiencies in communication among project developers. UML attempts to mitigate these shortcomings by attempting to unify the semantics where "by unifying the semantics and notation, they could bring some stability to the object-oriented marketplace, allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features." [20]

A request for proposal (RFP) issued by The Object Management Group in 1996, and by 1997 contributions from many leading software companies such as IBM, Microsoft, Hewlett-Packard, and Digital Equipment Corporation (DEC) produced UML 1.0. Within the year, UML 1.1 was released as a formal specification to improve the clarity of UML 1.0's specifications, and to allow new contributions from new OMG partners. UML 1.1 served as "a language for specifying, visualizing, constructing, and documenting the artifacts of software systems". [20]

Goals of UML

The first stated goal of UML 1.1 is to "provide users a ready-to-use, expressive visual modeling language (notation) so they can describe and exchange meaningful models." To accomplish this, UML 1.1 defined the following graphical diagrams as primary artifacts [23]:

- *Use Case Diagram*
- *Class Diagram*
- Behavior Diagrams
    - *Statechart Diagram*
    - *Activity Diagram*
    - Interaction Diagrams

    - *Sequence Diagram*
    - *Collaboration Diagram*
  - Implementation Diagrams
    - *Component Diagram*
    - *Deployment Diagram*

These eight diagram types are designed to provide sufficient coverage in communication of ideas and information as software and systems engineers work together to specify, construct, visualize, and document artifacts of a system—primarily software systems. More precisely, 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. It is important to note that the Statechart Diagram is based on Harel's work (remember, Harel is the author of the original higraph formalism).

Revisions of UML 1.1 continued through UML 1.4, which was published in 2001 to make minor improvements on previous versions. No significant changes were made to the number and type of primary artifacts in UML 1.4.

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.

Subsection 3.1.2: Weaknesses of UML 1.x

From a systems engineering perspective, UML 1.x provides little support for requirements definitions and traceability. Requirements Engineering is a crucial step (some would say the most crucial step) in the Systems Engineering Process, and effective requirements engineering allows for the creation, management, and allocation of requirements to systems, sub-systems, and components.

UML 1.4 only allows for requirements to be expressed as comments on Stereotypes; and the language also provides for traces. Together these features allow for very manual requirements management within a UML diagram. A major weakness is that all requirements are not pulled from a single requirements model as would be desired. In fact, the term "requirement(s) model" does not even appear in the UML 1.4 specification [18].

The absence of Flow Diagrams from this list is noteworthy. The flows in question could be flows of information, physical or electrical signals, or communications. During the early years of development, the UML authors did not believe flow diagrams fit into the Object Oriented paradigm that framed UML development.

The following quote from [5] 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…In addition, diagrams may represent different views on a system…There is no mechanism which defines the interconnections between 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.

Subsection 3.1.3: What is UML 2.0?

UML 2.0, formalized in 2005, made use of all existing UML 1 diagrams except the Collaboration Diagram, but added to and rearranged them to provide improved support for modeling of real-time systems [19]. The revised list of diagrams is as follows:

- Structure Diagrams
  - ***Class Diagram***
  - ***Component Diagram***
  - *Object Diagram*
  - *Composite Structure Diagram*
  - *Package Diagram*
  - ***Deployment Diagram***

- Behavior Diagrams
    - ***Activity Diagram***
    - ***Use Case Diagram***
    - ***State Machine Diagram***
    - Interaction Diagrams
        - ***Sequence Diagram***
        - ***Communications Diagram***
        - *Timing Diagram*
        - *Interaction Overview Diagram*

Diagram types written in bold font are carried over from UML 1.x. The diagram types written in italic font are new.

UML 2.0 made significant improvements in documenting flow of information in a system. By adding Communications, Timing, and Interaction Overview Diagrams, more types of communications and interactions (flows) could be shown.

Additionally, the new Parts, Ports, and Connectors [27] allow for a decomposition of systems into subsystems, components, parts, etc. This hierarchical representation is crucial to the modeling and evaluation of real-life systems.

Subsection 3.1.4: Weaknesses of UML 2.0

In UML 2.0, like UML 1.x, we still see little additional effort given to requirements modeling, functional allocation, or domain specific (customized) viewpoints. (This is done, in part, to keep the focus of UML remaining on software and real-time software systems.) Others, however, saw a need for better requirements modeling, and in [13] Letelier documents an entire Requirements Traceability metamodel. This metamodel works within the specifications of UML to not only show requirements traceability, but 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.

Finally, UML models tend to be most effective in the early stages of the Systems Engineering lifecycle—where engineers are organizing their ideas and are creating design options. Something that is all too often overlooked, and is not thoroughly addresses by UML, is Test Engineering. The process of developing Test Cases to validate and verify requirements, designs, and implementations is something that is on-going and iterative during Systems Engineering. UML does not provide adequate Test models.

Section 3.2: The Systems Modeling Language

Subsection 3.2.1: What is the Systems Modeling Language

While UML 1.x and UML 2 made significant positive advances towards a true Systems Engineering modeling language, many holes remained. Development on Systems Modeling Language (SysML) began in 2003, and in 2005 the alpha spec was published [25]. *(There was no shortage of solicited and unsolicited input to the SysML specification as various parties tried to improve—and tailor—some SysML specifications. See [3] for one of the more thoughtful examples.)*

SysML is a "domain specific language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities." [25]

From this description alone, we can see the intentions of the SysML working group to expand on UML 2.0. While UML focused on specifying, constructing, visualizing, and documenting artifacts of a (primarily software) system, SysML's focus shifts toward supporting the entire Systems Engineering lifecycle from specification to validation. SysML was written with non-software systems in mind.

As a result, SysML implements the following diagrams organized into three sections; diagrams for modeling system structure, for modeling system behavior, and those that cut across viewpoints [25]:

- Structure Diagrams
    - Block Diagram
        - ***Block Definition Diagram*** *(extends UML Class Diagram)*
        - ***Internal Block Diagram*** *(extends UML Composite Structure Diagram)*
    - Parametric Constraint Diagram
        - *Parametric Definition Diagram*
        - *Parametric Use Diagram*
- Behavior Diagrams
    - ***Activity Diagram*** *(extends UML Activity Diagram)*
    - ***Use Case Diagram***
    - ***State Machine Diagram***
    - ***Sequence Diagram***
- Cross-Cutting Diagram
    - *Allocation Diagram*
    - ***Package Diagram*** *(extends UML Package Diagram)*
    - *Requirement Diagram*

Again, re-used diagrams are written in bold font. The new Parametric diagram "follows the graphical conventions of a UML internal structure diagram showing a collaboration." [25]. Parametric constraints can be used in the ever-common trade-off study to show what happens to one (internal) characteristic of a Block, if another characteristic on another Block is changed. Parametric diagrams can not only show design specifiable characteristics of a Block, they can also be used to show real-world, domain level rules and constraints (such as engineering equations).

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

Strengths of SysML

Compared to UML, SysML offers the following new features:

- **Block Stereotypes.** The SysML Block Stereotype is based on the UML concept of composite structures. Blocks can have internal features (attributes, operations) and can own ports. An example of a hierarchy of system structure using blocks is shown below in Figure 13.

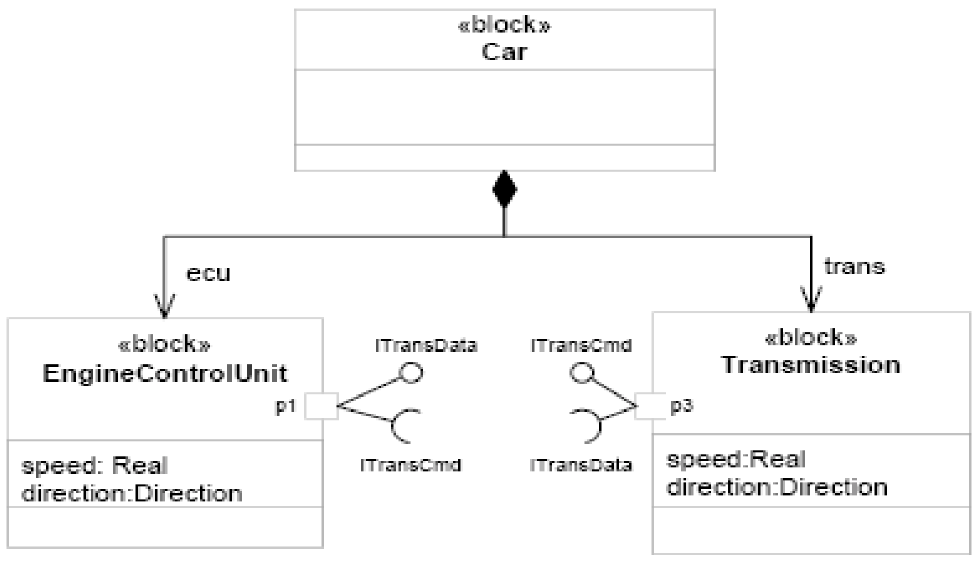


Figure 13: Example of Block Stereotype from SysML 1.0 Specification

The extension of UML Ports in SysML as FlowPorts provides a for more complete system model in which Blocks can be connected (physically and/or logically) to other Blocks. The abstract idea of a Block is perfect for initial Systems Engineering design phases when it is too early to specify the implementation features of a System such as specific parts and components.

- **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). In [13], Letelier only discusses allocation of requirements, so SysML makes a strong effort to acknowledge the importance of allocation in Systems Engineering. Unfortunately, the actual implementation in SysML is just a comment on a Block as shown in Figure 14 below.

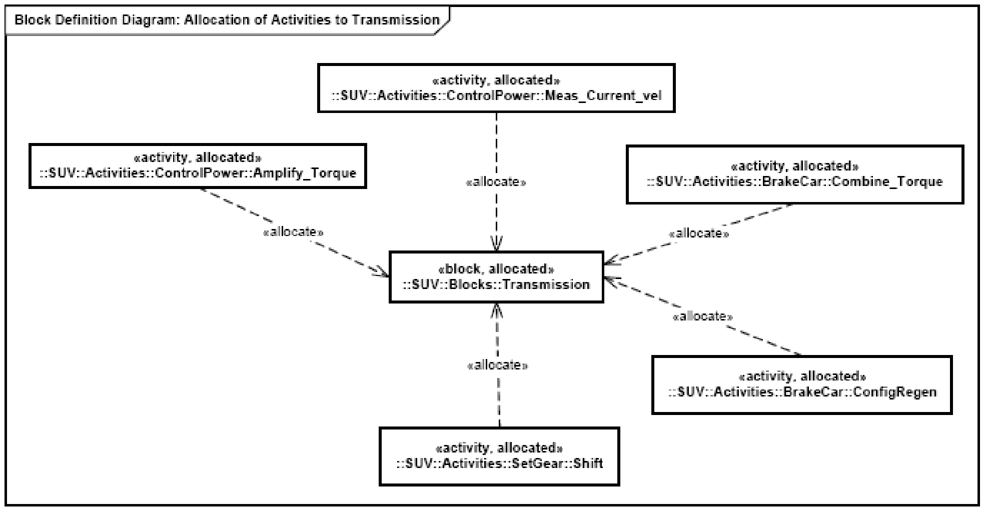

Block Definition Diagram: Allocation of Activities to Transmission
«activity, allocated» ::SUV::Activities::ControlPower::Meas_Current_vel
«activity, allocated» ::SUV::Activities::ControlPower::Amplify_Torque
«activity, allocated» ::SUV::Activities::BrakeCar::Combine_Torque
«allocate»
«allocate»
«allocate»
«block, allocated» ::SUV::Blocks::Transmission
«allocate»
«allocate»
«activity, allocated» ::SUV::Activities::BrakeCar::ConfigRegen
«activity, allocated» ::SUV::Activities::SetGear::Shift

Figure 14: Example of Allocations from SysML 1.0 Specification

- **Requirements Modeling.** SysML "provides modeling constructs to represent requirements and relate them to other modeling [system] elements." [25] SysML introduces an actual requirements node which contains information about requirements such as:

    - Unique Identifier
    - Requirement Source
    - Requirement Text
    - Verification Method

  These requirements nodes can be used in Block Definition Diagrams (UML version of a Class diagram) to show a hierarchy of requirements like in Figure 15 below. Requirements can also be mapped to other elements by derivation, verification, and satisfaction paths. For example, a diagram can show how a specific requirement is assigned to a component in the system structure.

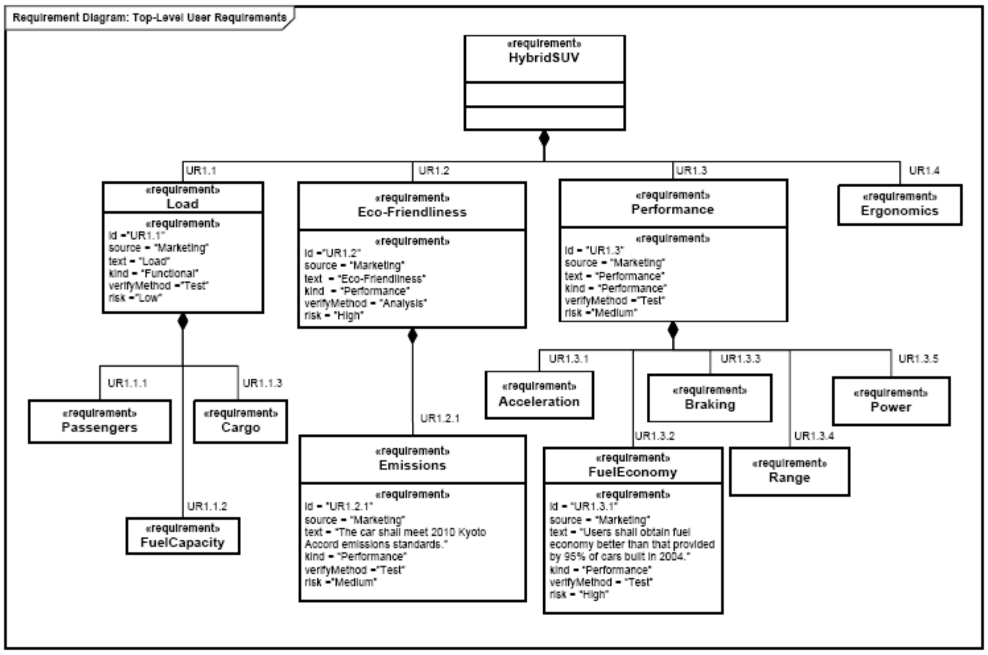


Figure 15: Example of Requirements Hierarchy from SysML 1.0 Specification

To summarize, by design, and compared to UML, SysML provides significantly improved support for the modeling of traditional Systems Engineering processes. There are, however, a few areas of weakness in the SysML alpha release.

Subsection 3.2.2: Weaknesses of SysML and Higraphs Areas of Improvement

We can see that SysML, as was intended, made significant improvements on UML to more fully model the traditional Systems Engineering processes. There are few areas of weakness, however, in the SysML alpha release. These are described below, along with the plan for how a higraph model would solve (or mitigate) such problems.

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. For instance, if a higraph requirements diagram and higraph structure diagram were created, these two diagrams could be linked to show how requirements are allocated to structure.

Support for Allocations

As we 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 seem to be no rules on allocations. In other words, how do we know if the <<allocate>> tag is correct? Though 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.

Forcing directional allocations (i.e. requirements to components, behaviors to components) to the lowest level possible would improve clarity of systems engineering decisions, and would allow for early validation of system correctness. Higraphs allow directional connectivity between all nodes in a higraph (or in higraphs as just mentioned)—from any level of the hierarchy, or any orthogonal region, to any other. Rules could be created that only allow certain types of edges (for example, allocations) to connect a requirement to a behavior, or connect a behavior to a function in a system structure component.

Moreover, enforcing rules for allocations (edge connectivity in higraphs) provides a basis for error checking within a system model. One of the best ways to perform error checking at the modeling phase of a Systems Engineering project, when SysML would first be used, is to thoroughly check for a complete traceability throughout the system. Traceability in SysML is allowed (though allocations), but is not forced. A language or modeling tool that forces the engineer to provide complete (and correct) traceability during the modeling process would be desirable.

By creating rules for edges in a higraph model, as has just been suggested, engineers would have a way to perform some level of validation on a higraph system model. 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).

To complicate matters, while SysML specifies hierarchical relationships among structure, behaviors (Black Box vs. White Box), and requirements, there is no clear definition of hierarchy among allocations. For instance, requirements can be allocated to sub-components, 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 this concept is easily dealt with by the realities of software development—the driving process behind UML, who's effects certainly carried over into SysML. Within software systems, and object oriented software systems specifically, inheritance and encapsulation can both be relied on to capture changes at lower levels (sub-classes) when changes to parent classes are made. However, in many other Systems 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 sub-components stay within specification (leads to low level change). *(Interestingly enough, in the SysML 0.90 DRAFT specification [24], there is a section devoted to nested connectors. While it's not too clear what the goal of these were, they could be used for hierarchical modeling ensuring traceability between levels of the system hierarchy.)*

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 he 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). 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. They are incapable of describing traceability relations between requirements, and structure and behavior entities.

A higraph model would always be defined by a mathematical formula. This would ensure that all relations between requirements, structure, and behavior entities are formalized, and these relationships must be honored for the model to be valid.

General Complexity of UML/SysML Specifications

Lastly, and perhaps most importantly, both the SysML and UML specifications are lengthy (some 200 and 700 pages respectively) and fairly complex. To be fair, they are both significant undertakings, and both do an exceptional job given the scope of the issues. However, in industry, there may not be enough time (or money) to educate project engineers on all of the language and syntax. Even if there were time and money, to be frank, many project engineers would not want to read and fully understand this specification. This could lead to project engineers not fully understanding the models that UML and SysML based tools will create of their systems, which will in turn lead to a lack of understanding of the system.

Higraphs, in contrast, are a general and very simple concept that is very easy to understand and use. Within the systems engineering domain, nodes represent various system pieces (requirements, structure, and behaviors) and edges show the relationships among those nodes. Node and edge definitions may be easily tailored to the needs of a specific application. Additionally, the higraph model can be created for (and extended to) any level of detail depending on the situation. As we will discuss in Section 5.2, existing software applications (Excel, Access databases) could be used to implement a Higraph model.

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 used as a system modeling language.

Chapter 4: Using Higraphs for Systems Modeling

The purpose of this chapter is to demonstrate 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. Each section presents a brief, real-world, example of how higraphs could be used to model requirements (power and physical), structure (an ATM), behavior (an automobile), and fragments of behavior assigned to structure (sequence diagrams). Section 4.4 describes how higraphs can link components from higraphs together to produce flows of design information generated during the system development. Real-world system level designs contain enormous amounts of information—the challenge of applying higraphs to real-world problems is discussed in Section 4.5.

Section 4.1: Requirements Modeling with Higraphs

At the beginning of any project, developing and documenting all requirements is critical to engineering success. As we will soon see, this task is complicated by the fact that requirements will often come from different domains and different stakeholders, from the technical and non-technical realm, and can be explicit or derived. A tenet of this thesis is that higraphs will allow all of these possibilities to be modeled formally and expressed visually.

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. Examples may be requirement number, requirement text, requirement priority, or requirement 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 Example I

Table 1 shows three requirements covering the electrical performance of an engineering system. Requirements 1.0 and 2.0 are explicit requirements; requirement 1.1 is derived from requirement 1.0.

Number	Requirement	Requirement Comment
1.0	The system must utilize less than 1500 Watts running on 220V power	Threshold Requirement
1.1	The system must consume less than 6.8 Amps	Derived Requirement from 1.0
2.0	The system must have battery backup for 60 minutes of operation.	Threshold Requirement

Table 1: System Requirements I Table

From the numbering scheme in the leftmost column of Table 1, it can be assumed that requirement 1.0 and 2.0 are at the same hierarchical "level", while requirement 1.1 falls underneath requirement 1.0 in the requirements hierarchy.

Figure 16 below shows the higraph implementation of Table 1

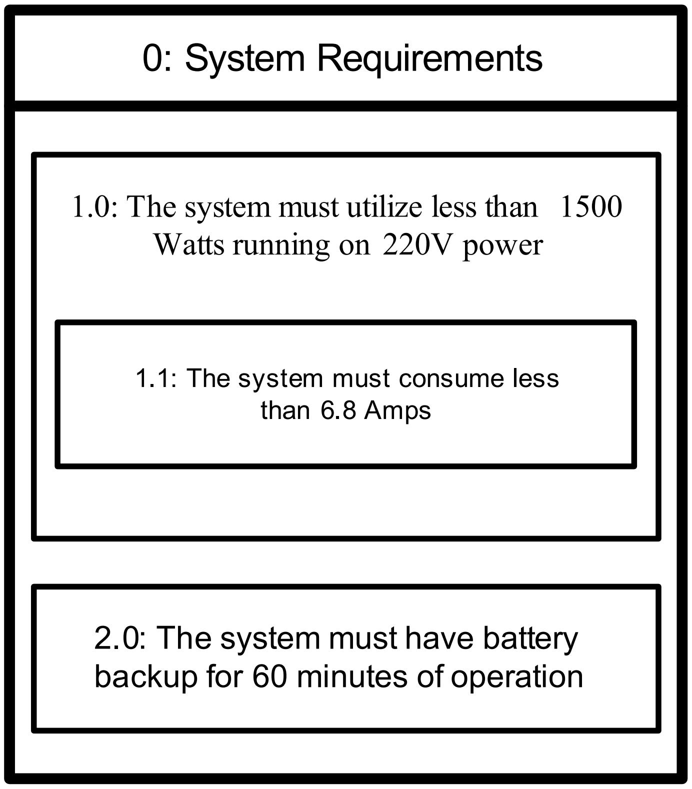

0: System Requirements
1.0: The system must utilize less than 1500 Watts running on 220V power
1.1: The system must consume less than 6.8 Amps
2.0: The system must have battery backup for 60 minutes of operation

Figure 16: System Requirements I Higraph

At this point it is worth noting that although edges are a fundamental part of the higraph formalism, they do not always have to be used. Indeed, as illustrated in Figure 16, the definition and representation of system requirements can be accomplished without explicit edges. The equivalent directed acyclic graph (DAG) representation of Figure 16 is shown below in Figure 17.

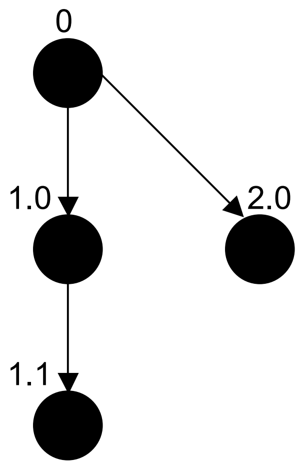


Figure 17: System Requirements I DAG

Logical Organization of Requirements

Requirements are commonly organized into tree ad graph hierarchies, especially for team based design. But this is not the only possibility. For example, another logical organization of requirements is by "domain." Orthogonality is a feature of higraphs that can be used to define and separate domains in order to logically group requirements. 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 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 the logical and visual separation of requirements from different domains.

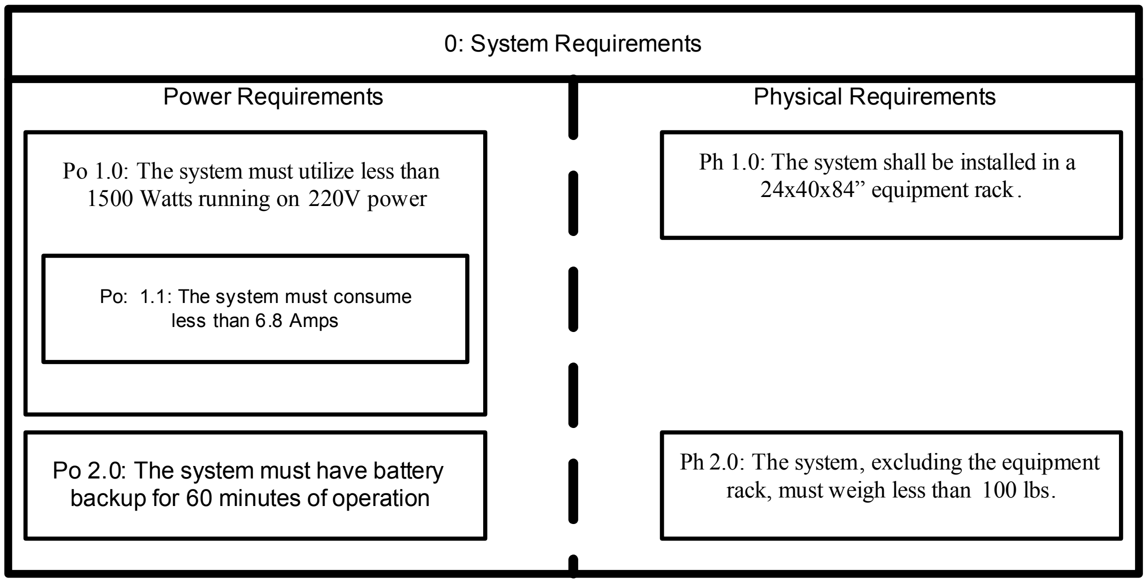
By using orthogonality to logically group nodes (requirements by domain in this case) we have to pay special attention to any edges crossing between orthogonal regions. That is, if nodes are separated—on purpose—through orthogonality, should they (could they) have relationships with other nodes from adjacent orthogonal regions.

Requirements Example I Continued

Now let's consider an example where the requirements emanate from multiple domains. The details are documented in Table 2, and are organized in a higraph in Figure 18 (with two orthogonal regions: Power Requirements and Physical Requirements).

Number	Requirement	Requirement Comment
Power 1.0	The system must utilize less than 1500 Watts running on 220V power	Threshold Requirement
Power 1.1	The system must consume less than 6.8 Amps	Derived Requirement from 1.0
Power 2.0	The system must have battery backup for 60 minutes of operation.	Threshold Requirement
Physical 1.0	The system shall be installed in a 24x40x84" equipment rack.	Threshold Requirement
Physical 2.0	The system, excluding the equipment rack, must weigh less than 100 lbs.	Threshold Requirement

Table 2: System Requirements I Table Continued



0: System Requirements
Power Requirements
Physical 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 Amps
Po 2.0: The system must have battery backup for 60 minutes of operation
Ph 1.0: The system shall be installed in a 24x40x84" equipment rack.
Ph 2.0: The system, excluding the equipment rack, must weigh less than 100 lbs.

Figure 18: System Requirements I Higraph Continued

The DAG representation of the Higraph in Figure 18 is shown in Figure 19.

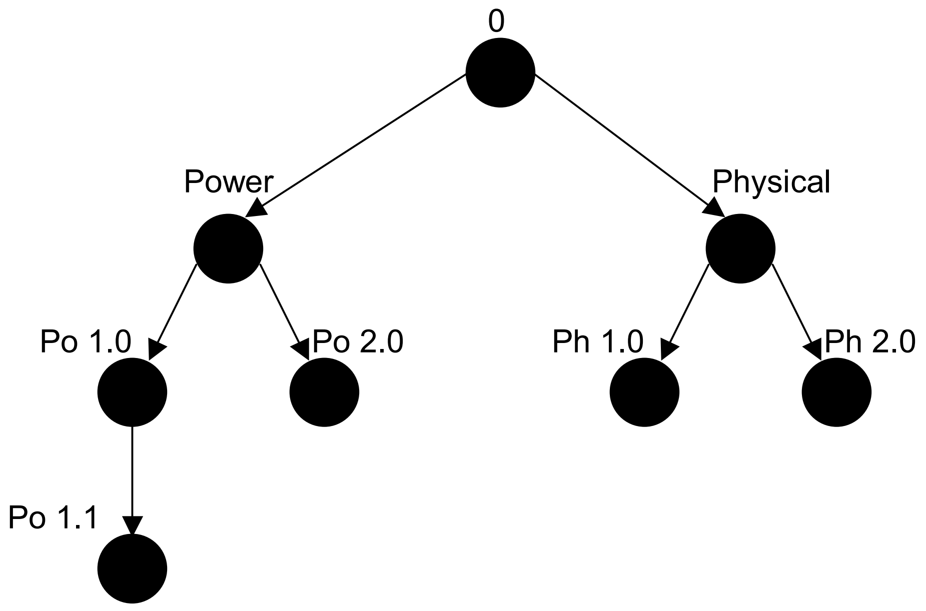


Figure 19: System Requirements I DAG Continued

As a side note, 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" in Figure 20 below. Harel creates this theory in [8].

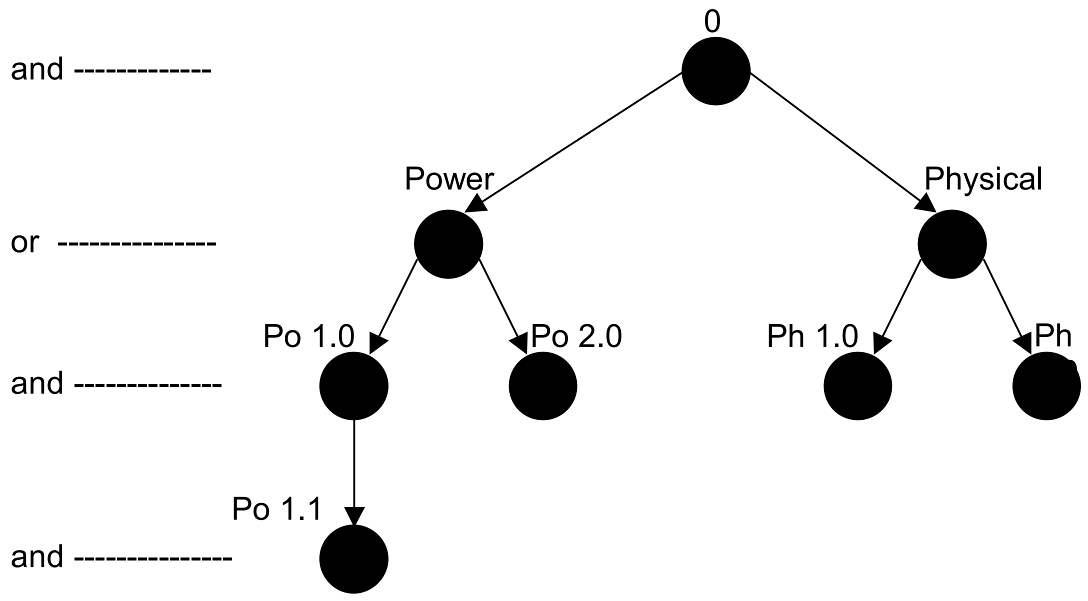


Figure 20: System Requirements II AND/OR DAG

Higraph Representation of a UML Case Diagram

A favorite diagram from UML and SysML that is often used in industry is the Use Case diagram. The Use Case Diagram will show what actions Actors (users, operators, maintainers, etc.) can perform using the system. Again, Higraphs can support Use Case modeling—taking everything from the Use Case Diagram (or Use Case table) including edges. By replacing the stick-figure icons representing Actors with the less aesthetically pleasing Higraph nodes, traditional Use Case Diagrams (like Figure 21 below) can be converted to Higraph Use Case Diagrams (like Figure 22 below).

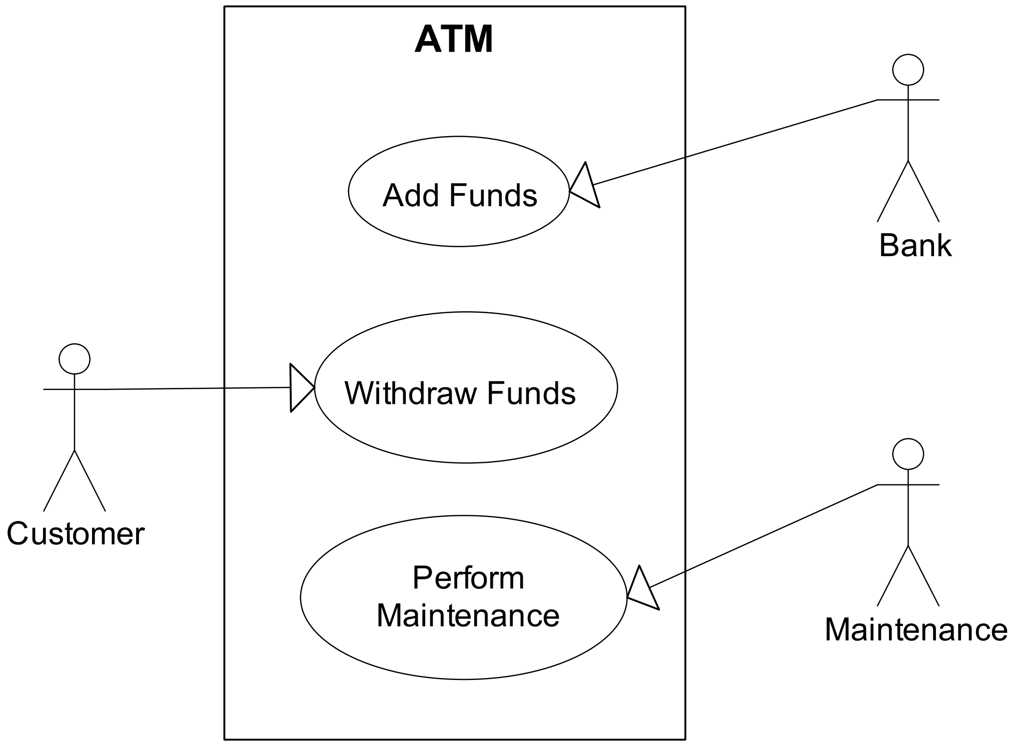


Figure 21: System Use Case Diagram I UML

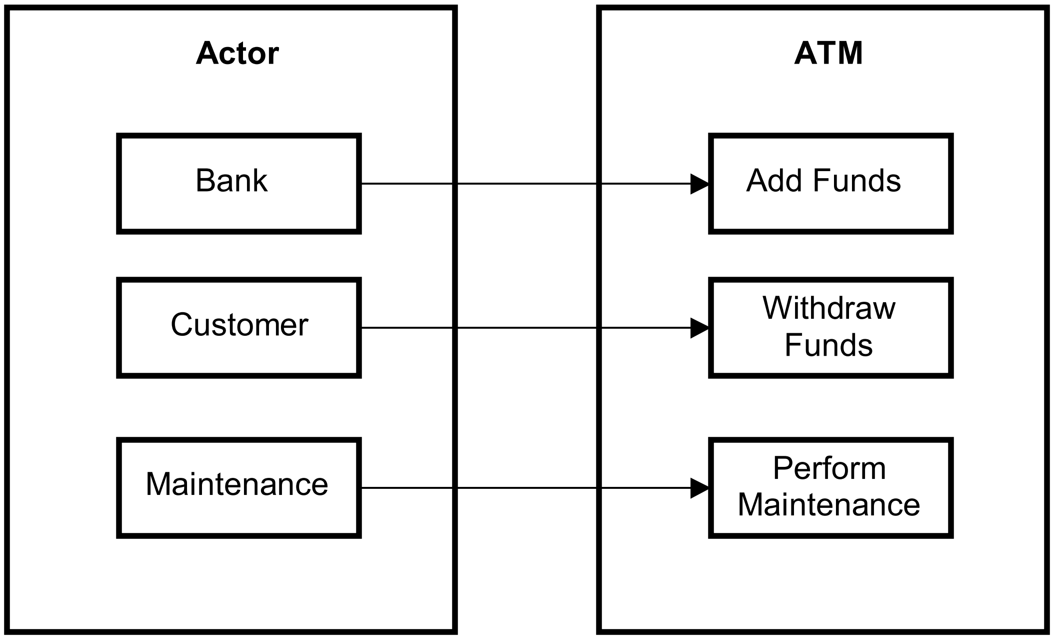


Figure 22: System Use Case Diagram I Higraph

Modeling Domain Requirements with Higraphs

Lastly, higraphs can be used to model requirements that belong to an entire domain—rules that apply to all systems of a certain type. The domain requirements may be a relevant portion of the existing principals of science such as electromagnetic fields equations for a communications system, or statics and dynamics principals for a bridge, etc. While the modeling of domain rules is established and mature, to do so with a higraph representation is new and novel. One of the key benefits in using higraphs is their support for [requirements] validation through traceability mechanisms implemented as higraph edges.

Section 4.2: Structure Modeling with Higraphs

System Structure modeling with higraphs is very similar (by design) to system structure modeling with UML and SysML. A 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. Within a "class" node, attributes and behaviors would be defined—each as a node enclosed within the "class" node. 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—"a part of" relationship that when the parent class is deleted, the sub class(es) will still exist. Composition, on the other hand, is a strong "has-a" relationship where if the parent class is deleted, the sub-class(es) will not exist. *(See [28] for a complete UML Glossary.)* Orthogonal regions can separate classes that aggregate or compose a parent class.

In a Higraph model of system structure, edges will be very significant (more so than in a Higraph requirements model). Edges will be used to show generalization, representing an “is a” relationship between classes. Inheritance of attributes and functions would follow these edges. Edges would also be used to show association, or other general relationships between classes.

Example UML Based Structure Model of an ATM

Figures 23 through 26 below employ UML class diagrams to show the structure of an ATM (adapted from [2]). Figure 23 shows the top level structure: The ATM is composed of hardware and software classes. Certain attributes and functions are also defined here (attributes and functions that will be inherited by all sub-classes shown in the subsequent UML diagrams).

Since 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. Figures 64 and 65 show a more traditional view that is purely structural in nature.

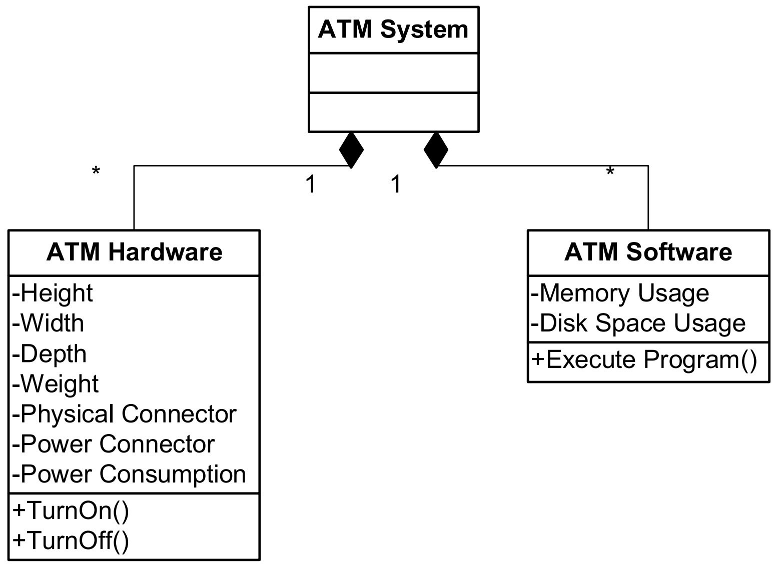


Figure 23: UML System Structure Diagram Top Level

Additional details of the class hierarchy are shown in Figures 24 through 26. Nodes at the bottom of the diagram (at the end of the open, unidirectional arrow) are generalizations of the ATM Hardware parent class (or ATM Software class parent class in Figure 26). 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.

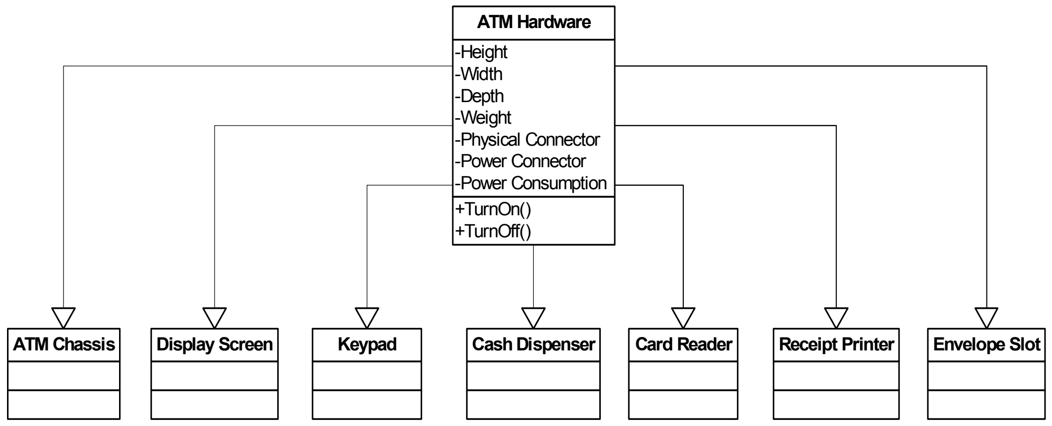


Figure 24: UML System Structure Diagram Aggregation

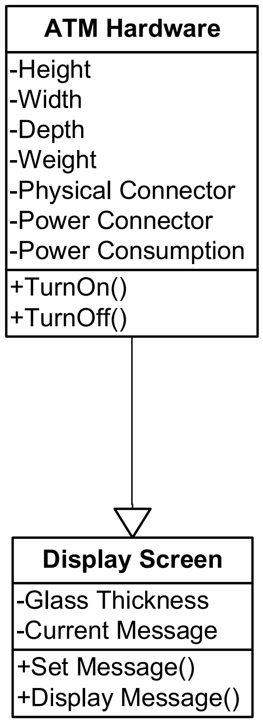


Figure 25: UML System Structure Diagram Hardware Component

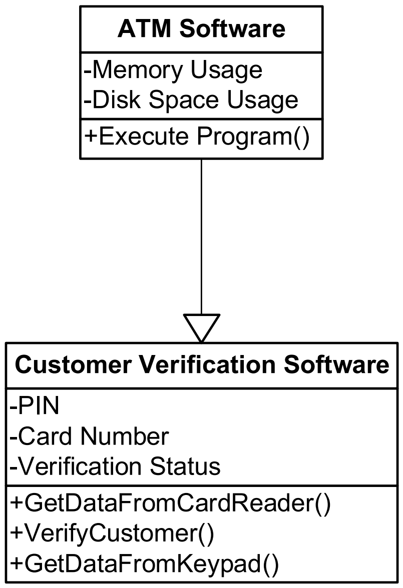


Figure 26: UML System Structure Diagram Software Component

Following the guidelines described above, we can convert the UML diagrams in Figures 23 through 26 to Higraphs.

Example Equivalent Higraph Based Structure Model of an ATM

Figure 27 shows the top level Higraph representation of the ATM structure. The composition relationship between the ATM parent class is shown using a hierarchical layout of nodes, and orthogonal regions for Hardware and Software nodes (classes).

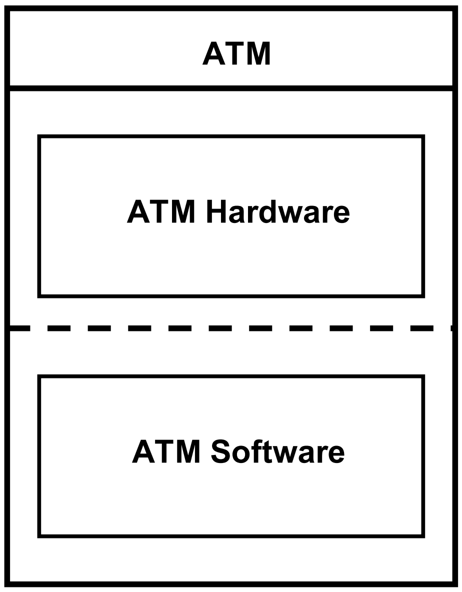


Figure 27: Higraph System Structure Diagram ATM Top Level I

Including more detail, Figure 28 below shows the same information included in the UML diagram shown in Figure 23. In this case, attributes and functions are shown in orthogonal regions in the class (node) to which they belong. By design, the use of orthogonality to partition attributes and functions is similar to the use of orthogonality to show a composition relationship. These attributes and functions exist only when their "parent" class exists.

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.

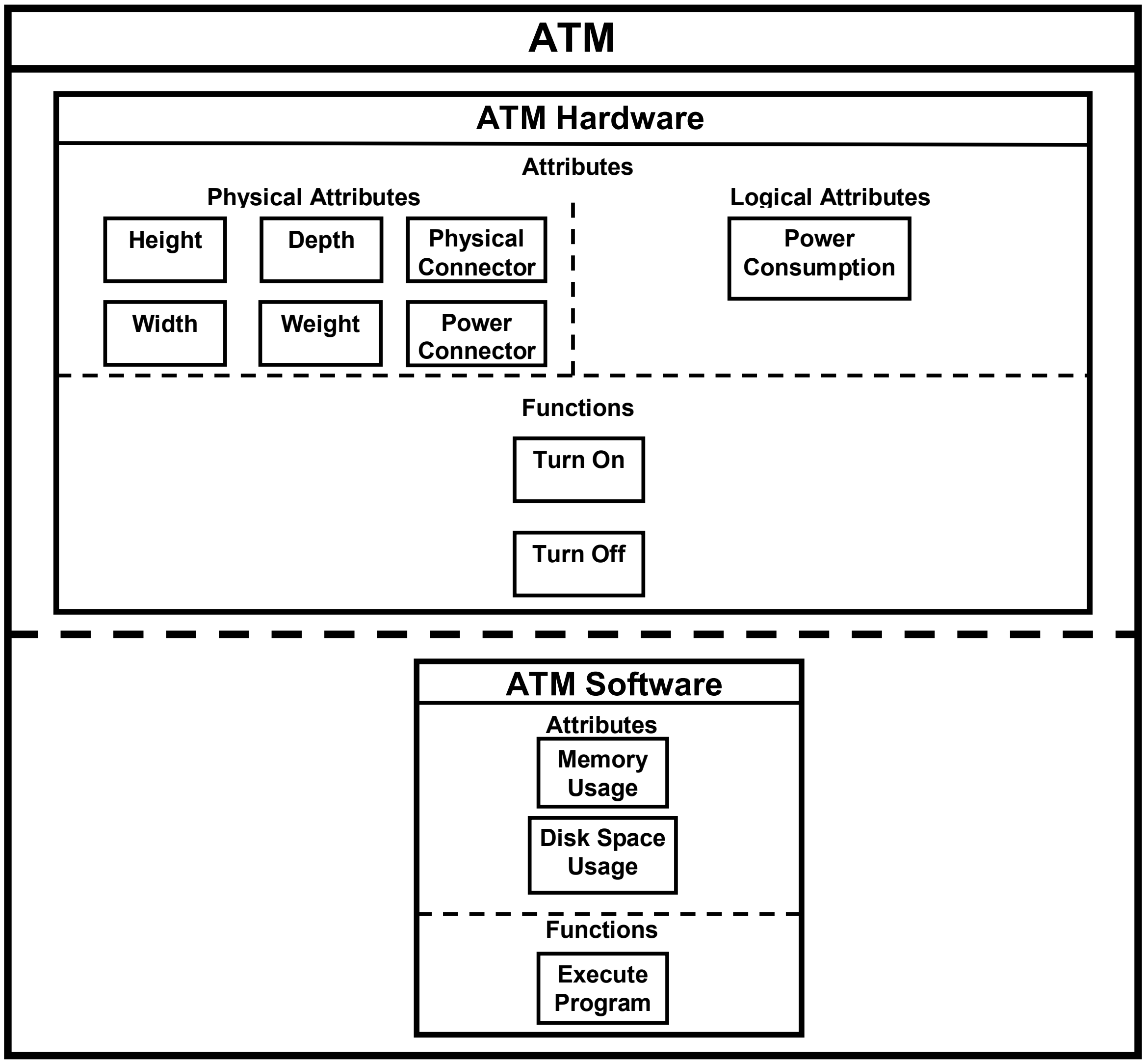


Figure 28: Higraph System Structure Diagram ATM Top Level II

Figure 29 below shows the generalizations of the ATM hardware and ATM software classes. Remember, the edges that represent generalization include inheritance from the parent classes. In this case, all generalizations of the ATM Hardware class include the attributes and behaviors from Figure 28 above. Likewise with the ATM Software class. Note also the multiplicity can be shown by adding allowed values to either end point of the edges in Figure 29 below.

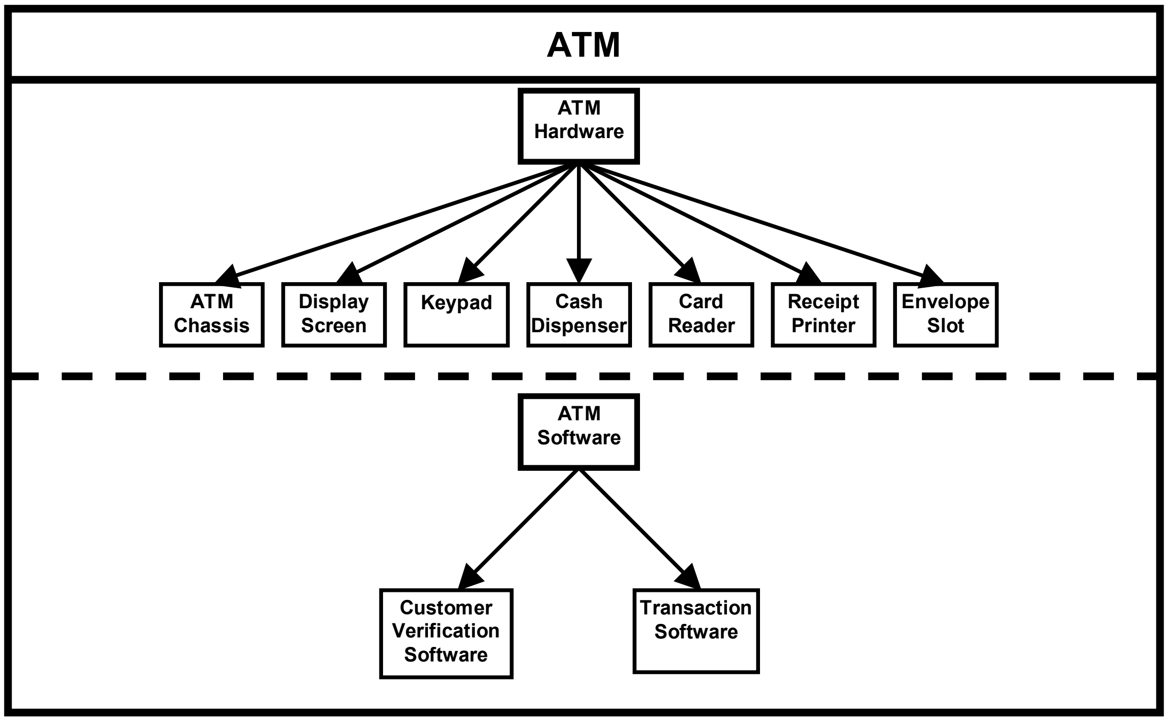


Figure 29: Higraph System Structure Diagram ATM Top Level Inheritance

With the Higraphs, varying amount of details can be shown. The two figures below, Figure 30 and Figure 31, show detailed definitions of classes in the ATM Structure model (and represent the same data from Figures 25 and 26, respectively). Again, the edges represent generalization (and inheritance) within these Higraphs.

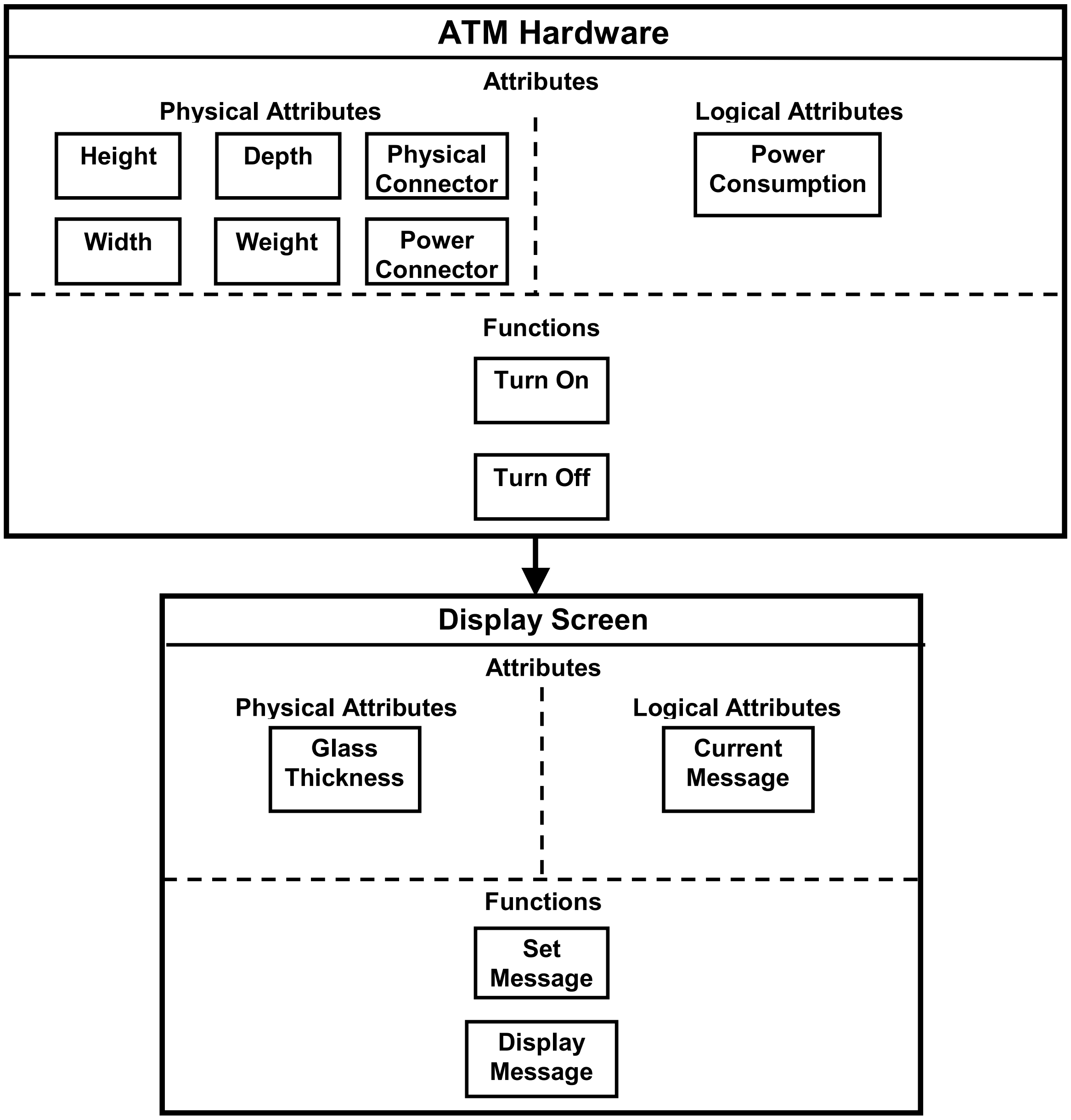


Figure 30: Higraph System Structure Diagram ATM Hardware Inheritance

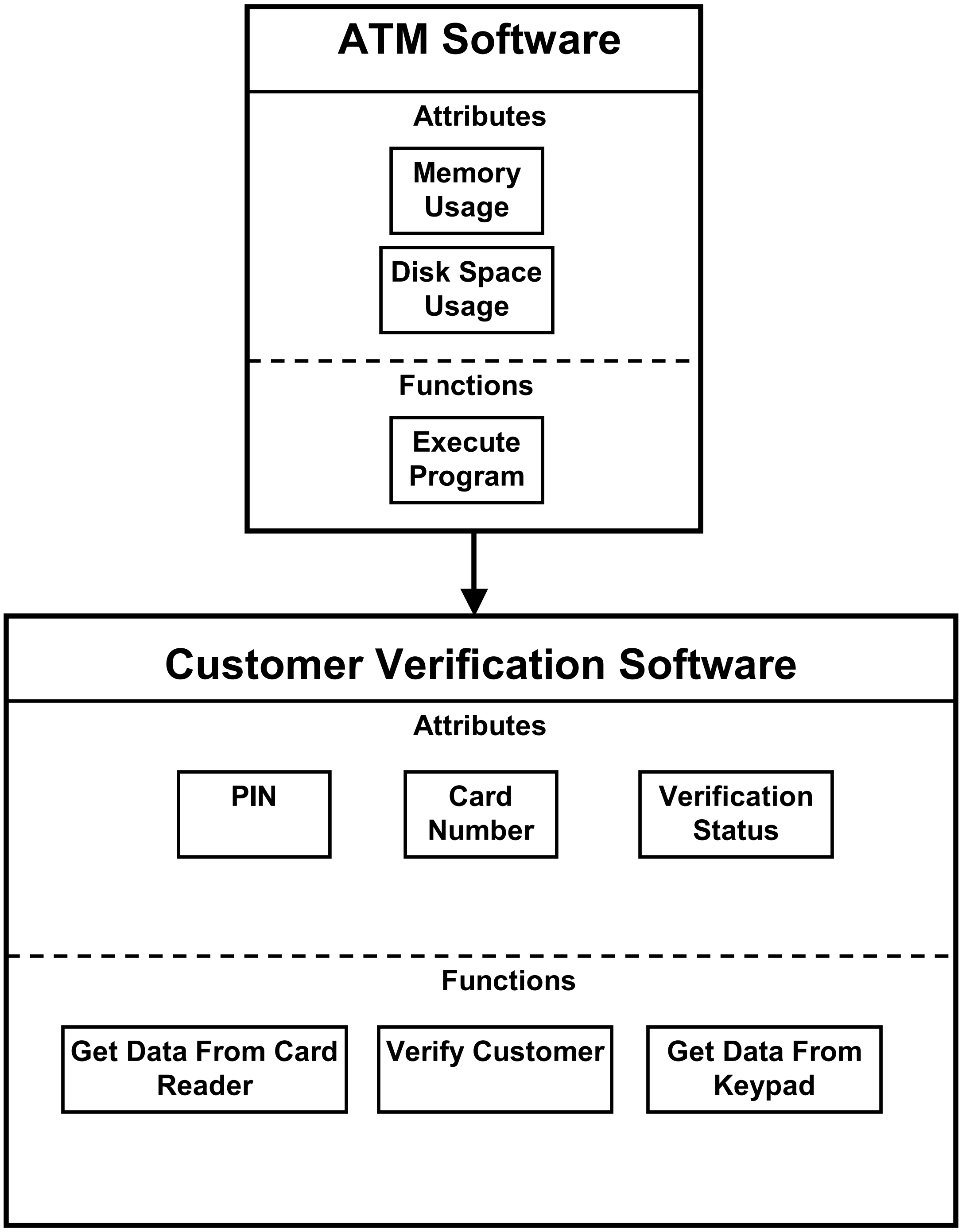


Figure 31: Higraph System Structure Diagram ATM Software Inheritance

In addition, Higraph structure models could also show all physical connections in a system using edges defining a specific (physical) association between two nodes.

Modeling Domain Structure with Higraphs

Again, like a Higraph requirements model, Higraphs can be used to model a Domain structure. Domain structure could represent fundamental rules that apply to a system's structure. For example, if modeling a new aircraft, a model of the Domain structure may show that two wings are required. The Higraph structure model should show two wings in the system design (unless there was to be some sort of change to the domain structure through a particularly innovative design). Another possible use for the domain structure model could be to represent available inventory—components that are available to be included in the system structure. This sort of bounded inventory is common in real-world manufacturing efforts as parts may only be available from selected suppliers.

Additionally, it is also worth reiterating that modeling domain structure and system structure with higraphs allows for a certain level of validation through traceability using higraph edges between the two models.

Section 4.3: Behavior Modeling with Higraphs

Higraphs have distinct advantages over traditional behavioral modeling formalisms such as state diagrams. As Harel points out, four drawbacks to the traditional state diagrams are [9]:

1. State diagrams are "flat", and thus do not support decomposition to various levels of detail.

2. State diagrams are uneconomical when it comes to a single transition (i.e. a high level interrupt) that affects a large number of states. In this case, a transition (or edge) must be drawn from each state.
3. In a traditional sense, when state diagrams show each state as being a “snapshot” of the system at a single point in time—capturing the “settings” and “configurations” of all components in the system to represent a specific state—as components are added to the system, the number of states increases exponentially.
4. State diagrams are inherently sequential in nature, and do not represent concurrency well.

Higraphs, in contrast, offer the following improvements to system [behavior] modeling:

1. Higraphs, being hierarchical in nature, allow for higher or lower levels of detail to be shown as needed and as appropriate. Indeed, support for hierarchical representations of systems means that higraphs can be used for top-down and bottom-up modeling of a system behavior.
2. Due to the topological grouping of nodes (states) in a Higraph, a single edge can represent a transition that affects any number of states.
3. 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. Orthogonal states provide a natural mechanism for modeling of systems that contain concurrent sub-system/component behaviors.

As a matter of fact, statecharts really are the basis for Higraphs [9].

For a complete Higraph view of system behavior, the Higraph components might be utilized as follows (for a system with a defined structure):

- Nodes are allowed system states. These states correspond to specific values of attributes defined within the nodes in the system structure model. By paying attention to the grouping of these states, the behavior model can remain in proportion to the size of the system structure model.
- Hierarchy represents varying levels of detail for system behaviors. High level behaviors (Plane::Fly, Car::Drive) can be made up of lower level activities.
- Orthogonality would be used to represent concurrency (simultaneous behaviors)
- Edges are events, internal or external, that cause the system to change states. Internal events should 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).

Example Higraph Based Behavior Model of Concurrent Features in an Automobile

The following diagrams, adapted from [1], are used to show a Higraph representation of a portion of a system behavior. Modern automobiles contain many electro-mechanical systems that, for the most part, have concurrent behaviors. As a case in point, Figure 32 below shows a portion of the system behavior of an automobile (certainly more behaviors could be added).

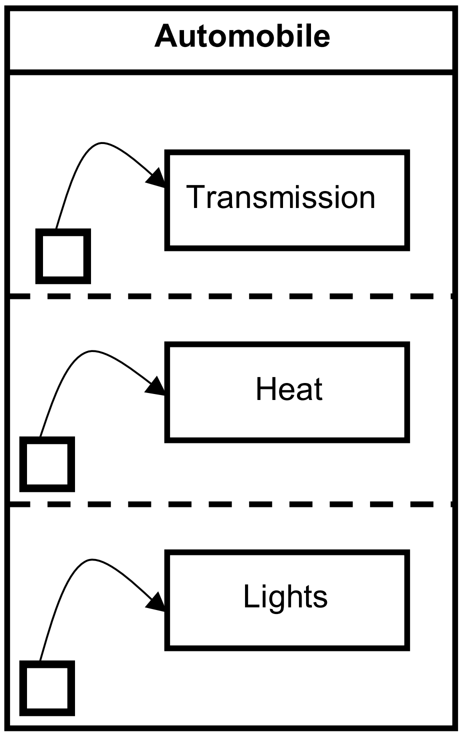


Figure 32: Higraph System Behavior Diagram Automobile Top Level I

In this figure, there are three orthogonal regions: Transmission, Heat, and Lights. Each region corresponds to a behavior (i.e. the behavior of the transmission, the behavior of the heating system, and the behavior of the car's lights). Only the top level of the behavior hierarchy is shown here. The transmission, heat, and lights can all be used concurrently (we would hope!). As such, each orthogonal region has a distinct initial state (the three empty blocks).

As we expand on the hierarchy of Figure 32 below in Figure 33, more details are added to the behaviors.

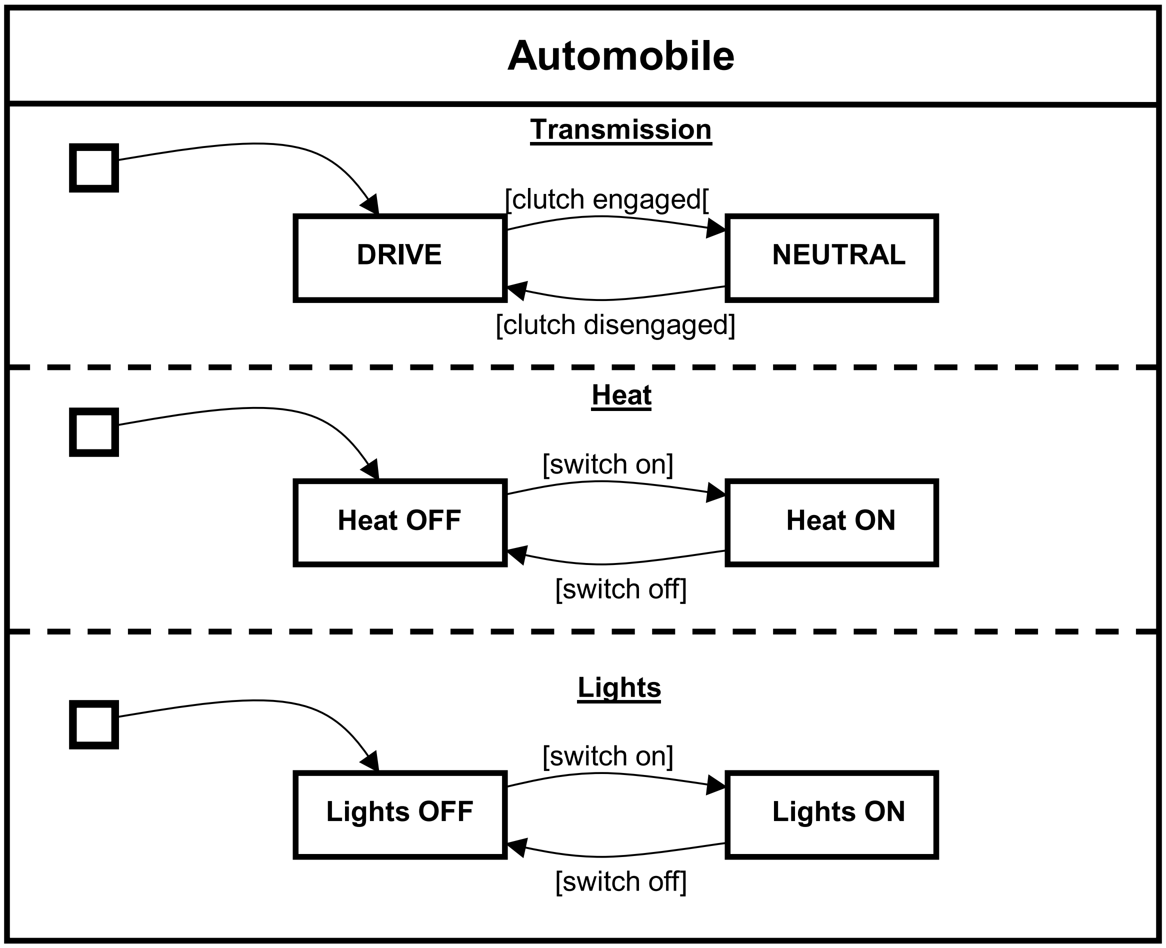


Figure 33: Higraph System Behavior Diagram Automobile Top Level II

The same three orthogonal regions are shown, but more detail is provided for each behavior. Remember from our guidelines above, the nodes here represent system states. That is, a set of specific values assigned to system component attributes. For example, the "Heat ON" state would correspond to the car's heater (component) running (heater attribute value), and the switch to activate the heating (component) set to on (heater switch function). In all three regions above, the events that trigger system states to change from drive to neutral, and off to on, are caused by external events (represented by the edges between the states).

One last level of hierarchy is shown in Figure 34 below.

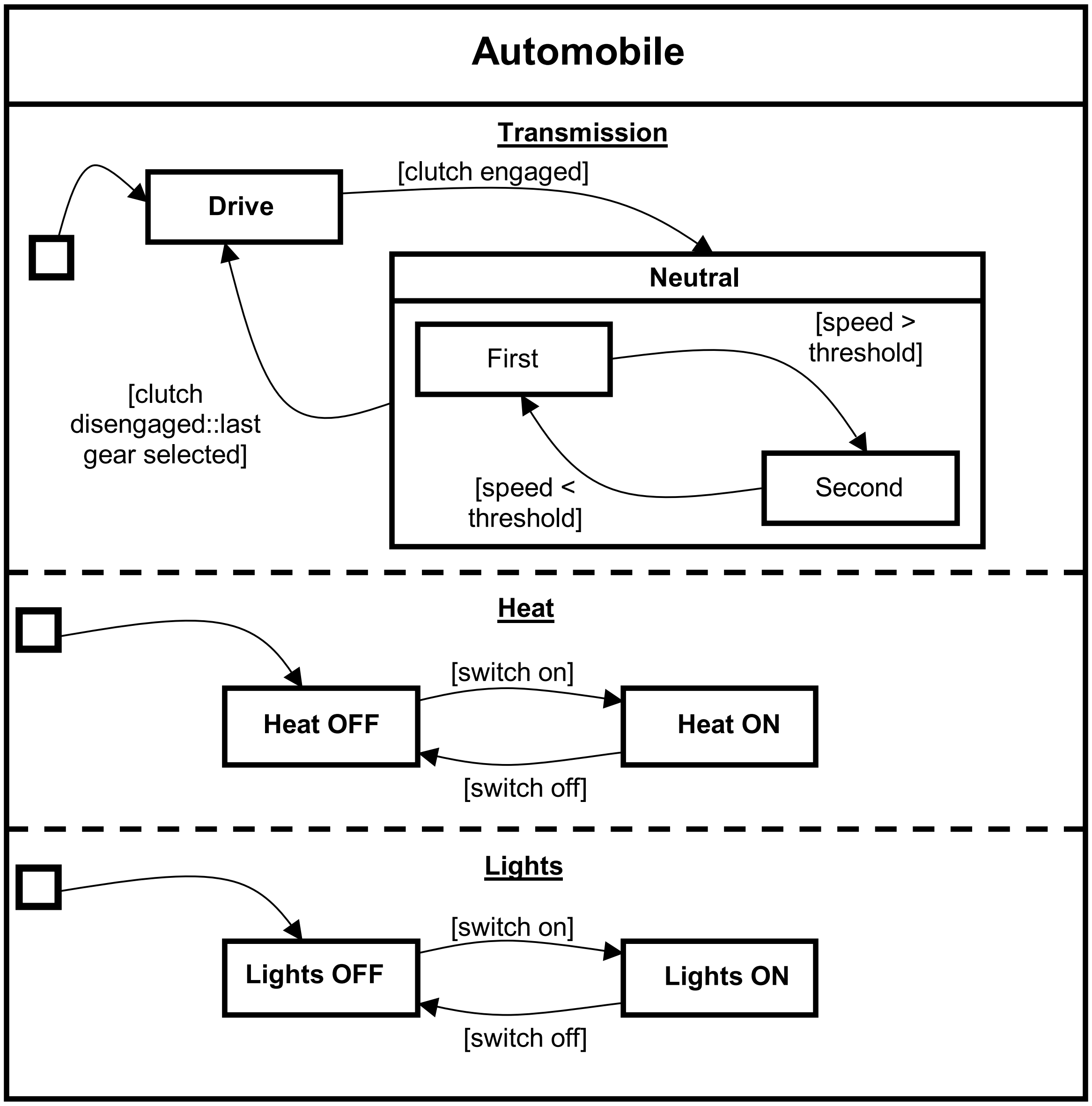


Figure 34: Higraph System Behavior Diagram Automobile Top Level III

We see more details about the allowed system states inside of the "Neutral" state. (Of course, this state could be expanded to show all gears.) Additionally, we have added a value to the edge leading from "Neutral to "Drive." This shows that the value of the "last gear selected" (what position what the shifter left in?) being passed to the "Drive" state.

Modeling Behaviors in the Absence of a System Structure

For a system whose structure has yet to be defined, Higraphs are flexible enough that they could still be used to model system behaviors. In this case, Higraphs components might be used as follows:

- Nodes are desired behaviors. These must correspond to functions that will be allocated to pieces of the system structure.
- Hierarchy still represents varying levels of detail for system behaviors. High level behaviors can be made up of lower level activities.
- Orthogonality would still be used to represent concurrent behaviors.
- Edges are transitions, based on the output of the functions that change current "state" of the system. These states must later correspond to attributes that will be defined in pieces of the system structure.

Additional ways to Represent System Behaviors using Higraphs

Higraphs can, of course, be used to show information found in other system behavior formalisms. Activity diagrams, with their activities (nodes) and transitions (edges) can easily be modeled as a Higraph. Decision elements would be supported by a specific type of node, and parallel behaviors would be supported by orthogonally divided activities. A simple UML example from the automotive domain is shown below in Figure 35:

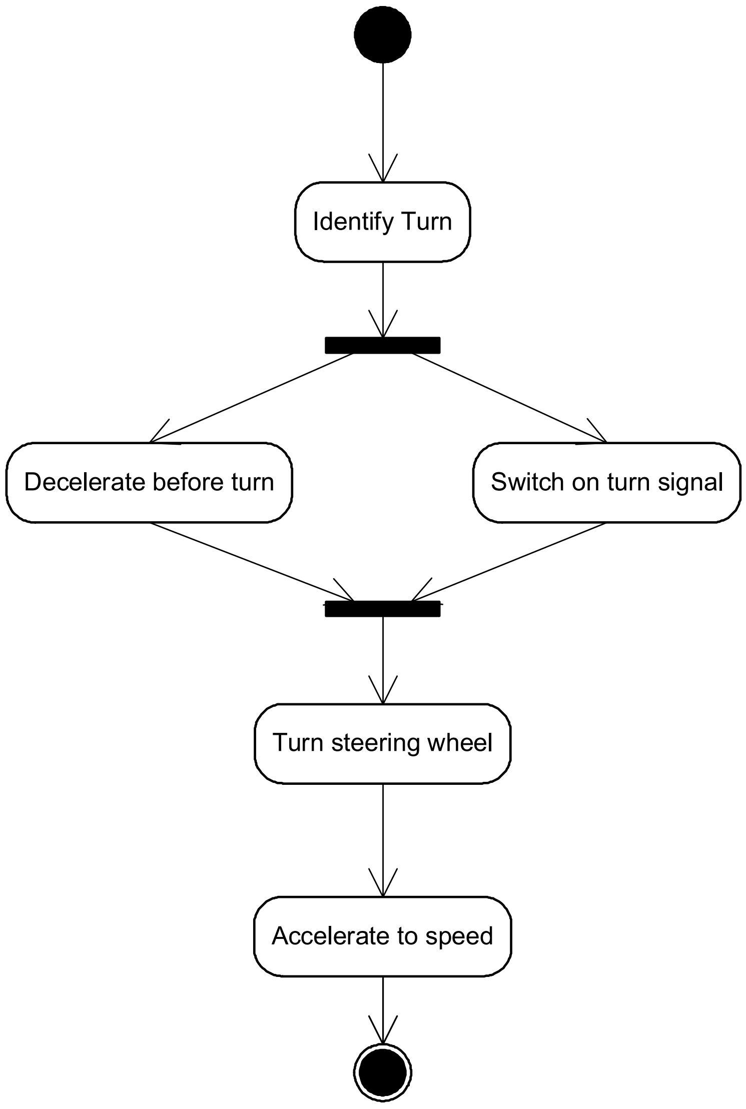


Figure 35: UML Activity Diagram for Turning Car

Figure 35 could be represented as Figure 36 in Higraph form.

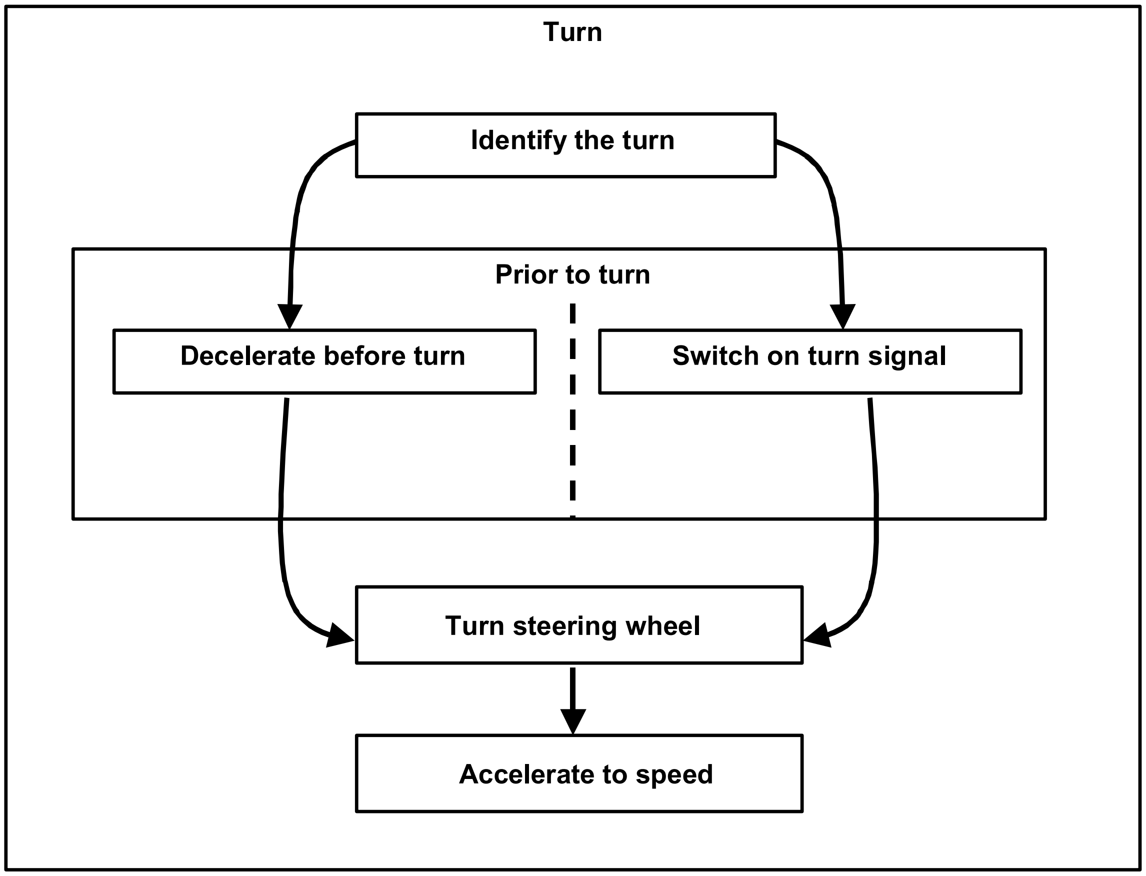


Figure 36: Higraph Representation of UML Activity Diagram from Figure 35

Likewise, Sequence Diagrams which show a sequence of events over time, can be modeled using higraphs. A UML Sequence diagram representing the sequence of events to enter a car is shown below in Figure 37.

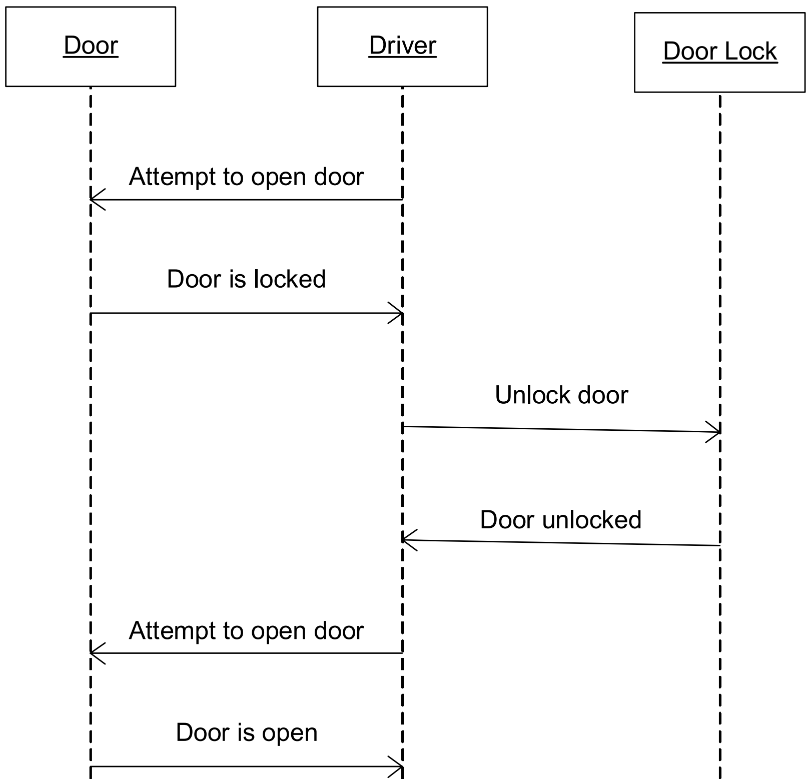


Figure 37: UML Sequence Diagram: Car Entry

The UML sequence diagram in Figure 37 can be shown as a higraph diagram in Figure 38. To do this, we have adapted a concept described in [16]. Note that messages (edges) originate from traditional structure object nodes (driver, door, door lock), but they must pass through a "time" node (with an attribute counting time) before arriving at another structure node.

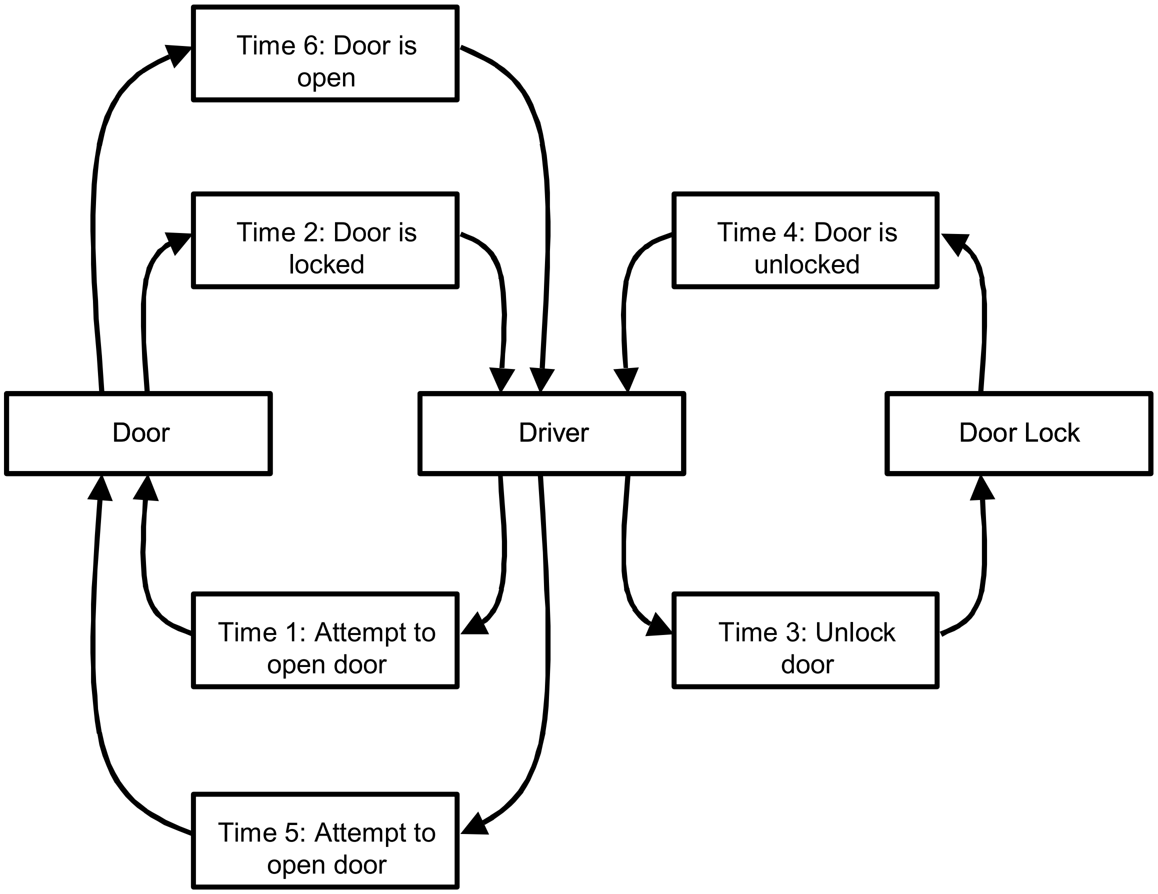


Figure 38: Higraph Representation of UML Sequence Diagram from Figure 37

It therefore, seems reasonable that in the long-term, higraph representations can compliment, and perhaps even co-exist, with UML and SysML representations of systems.

Modeling Domain Behaviors with Higraphs

Lastly, like requirements and structure modeling described earlier, higraphs can be used to model domain behaviors. A large range of domain behaviors are possible, be it the natural sequence of time (seconds, minutes, hours) that is applicable to a real-time system, the tidal patterns (for a maritime system), and so forth. The point here is not that domain behaviors are unique to higraphs—they are not. What higraphs offer is the ability to connect "domain models" of behavior to viewpoints of the system design. This, in turn, allows for early validation of system behavior models.

Section 4.4: System Level Modeling and Connectivity with Higraphs

To this point, we have introduced the Higraph formalism, discussed its fundamental parameters, discussed existing system modeling languages, and discussed how Higraphs could be used for system requirements, structure, and behavior modeling. In some cases, a higraph formalism provides equivalent functionality to diagrams in UML/SysML (e.g., Use Case Diagrams, Statecharts). Now we intend to show how individual higraphs can be linked, thereby creating a framework for a true, unified, system level model.

The primary strength of a higraph representation of system requirements, structure, and behavior is explicit support for traceability (via edges) between all nodes in the separate higraphs. Although system requirements may be defined in one higraph, system structure defined in another, and system behavior defined in yet one last higraph, all three higraphs can be connected (made into one large higraph) by creating edges between the nodes in each, thus creating a true system model.

For instance, one requirement node may be connected to a structure node (a system component) indicating that this piece of the system structure satisfies that requirement. Additionally, a requirement node may be connected to a behavior node (a system function) indicating that this system behavior satisfies that requirement.

The mapping could continue, with behaviors being connected to pieces of the system structure to complete a functional allocation—that is, assigning a behavior to a specific system component.

Example Allocation of Requirements to Structure and Behavior

Figure 39 shows how one higraph can link together representations for system requirements, system structure, and system behavior. Each of these concerns (or products of system development) can be represented by their own, individual, higraphs (as shown in Sections 4.1 through 4.4). For this example, system requirements are contained in a higraph containing 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 would 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 to pass through a node in the interface requirements area to go from physical to functional (or vice versa).

This is just one possible representation for system level requirements. Physical, functional, and interface requirements may or may not be present in each system.

System structure is defined inside the system structure higraph, and system behavior is defined inside the system behavior higraph. Edges connecting the three higraphs show what pieces of the system structure *satisfy* specific physical requirements, and what system behaviors *satisfy* specific functional requirements. (Or, depending on how the design was accomplished, what system requirements *specify* specific system components, and what system requirements *specify* specific system behaviors.) System behaviors can then be *allocated to* components of the system structure to complete a functional allocation.

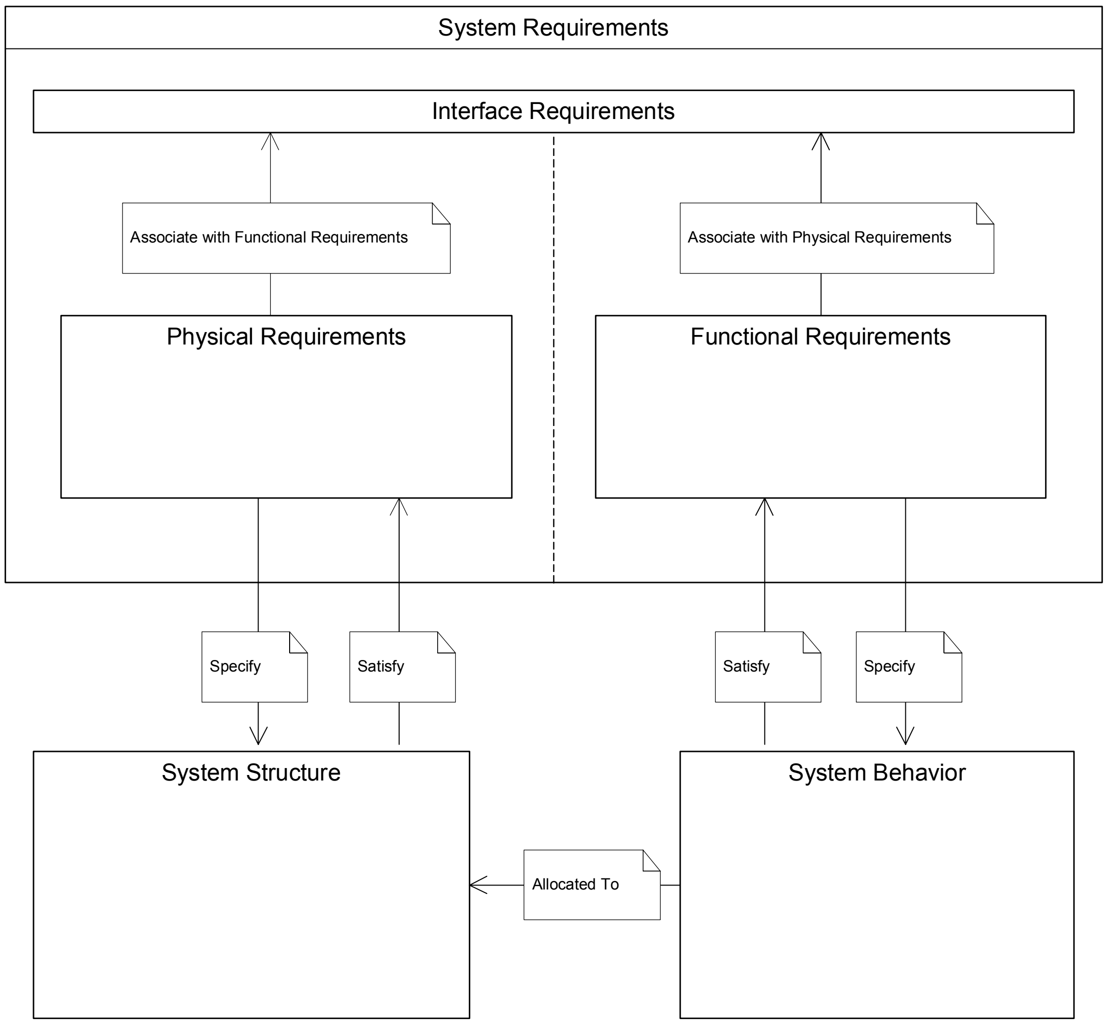


Figure 39: Higraph Based System Model

It is important to reiterate the hierarchical nature of Higraphs at this point. The diagram above shows the highest level of a system model, but within each node (System Requirements, System Structure, System Behavior) multiple levels of requirements, structure, and behaviors can exist as shown in the diagrams from earlier in this chapter. Edges between the higraphs can flow from any node at any level to any node at any level as required—or flow through an interface requirements node in the case of relationships between physical and functional requirements.

Example Domain Level Modeling

Higraphs can be made (or may already exist from previous engineering projects) for domain requirements (or domain rules), domain structure, and domain behavior. Connections between these higraphs show how the domain structure *complies with* domain rules (physical realities), and how domain behaviors *comply with* domain rules (functional realities). As a case in point, Figure 40 below shows the highest level representation of a higraph based domain level model.

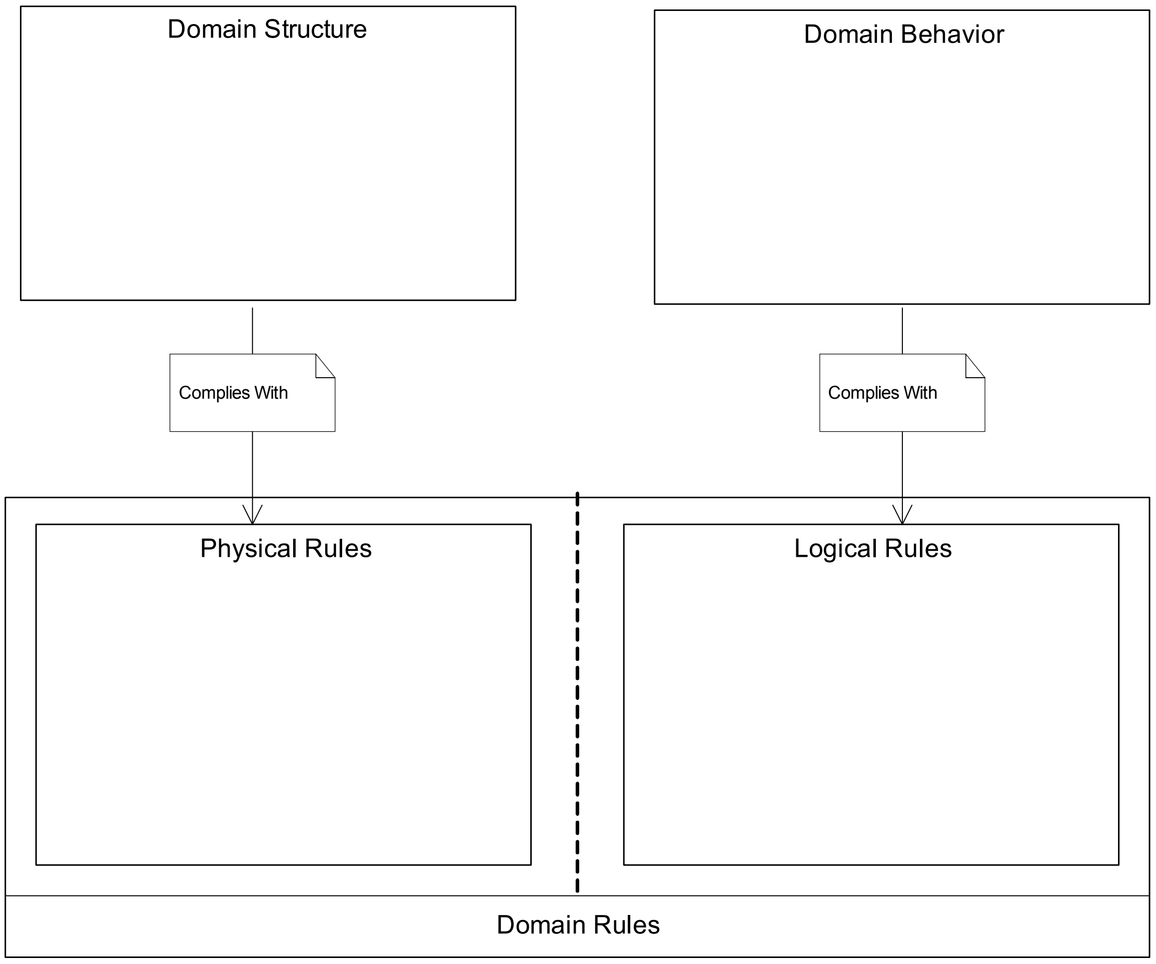


Figure 40: Higraph Based Domain Model

Section 4.5: Higraph Deficiencies

Although Higraphs are fundamentally sound tools for modeling due to their graphical (in a mathematical sense) nature and flexibility, there is one significant drawback in the pathway from definition to implementation. This is the complex process of arranging the nodes and edges in a visual layout that maximizes communication of information to an end-user. The problem can be difficult, even a small sized higraph, and becomes critical in a large scale (industrial) system.

In a large system of many components, behaviors, and domains, there most likely exists an incredible amount of overlap among the nodes that make up a higraph model of such a system. *(As you've probably noticed, most of the higraph examples presented to this point have just been very simple pieces of a system.)*

Example of the Visual Complexities Associated with a Higraph

An example of how the visual representation of a higraph could quickly move towards an unusable graphic is shown in the following example:

In a higraph model of system requirements, you could have nodes for each requirement domain (physical requirement, electrical requirements, and functional requirements). Within each domain you will have some high-level requirements—themselves represented by a node. Finally, each high-level requirement could be made up of sub-nodes that are its derived requirements. The resulting visualization can go from simple to complex for even the smallest of systems.

Figures 41 through 43 show the progression of higraph modeling of domain requirements, high-level requirements, and derived requirements for a simple point-to-point communications system (telephones connected via wire through some sort of switch).

Figure 41 below shows the highest level view of system requirements broken down by domain.

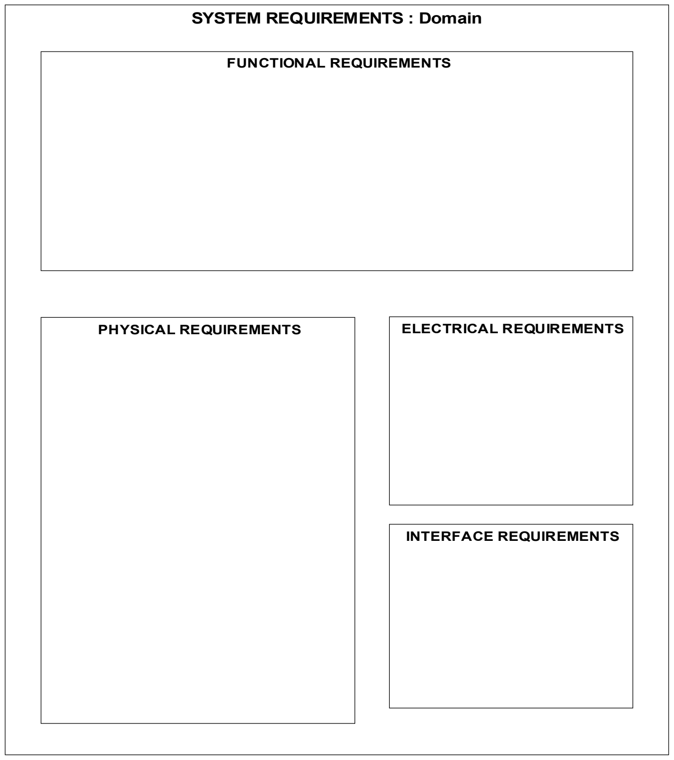


Figure 41: Higraph model of Domain Requirements.

Figure 42 below shows a view of system requirements broken down by domain, but includes the high-level requirements within each domain. From this figure we can see that a functional requirement of this system is to transmit and receive voice. This Higraph also indicates there will be some physical requirements associated with system components: end points (i.e. telephones) and a switch.

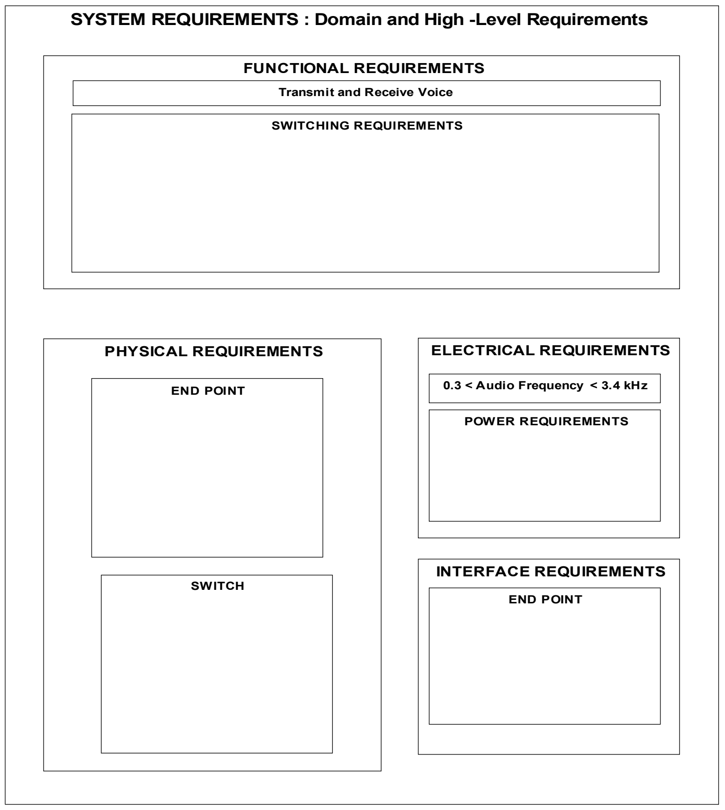


Figure 42: Higraph model of Domain and High-Level Requirements

Finally, Figure 43 below shows a view of all low-level/derived requirements within each domain and coming from high-level requirements. You can see, however, that this higraph has quickly become visually complex for even the smallest, and simplest of systems.

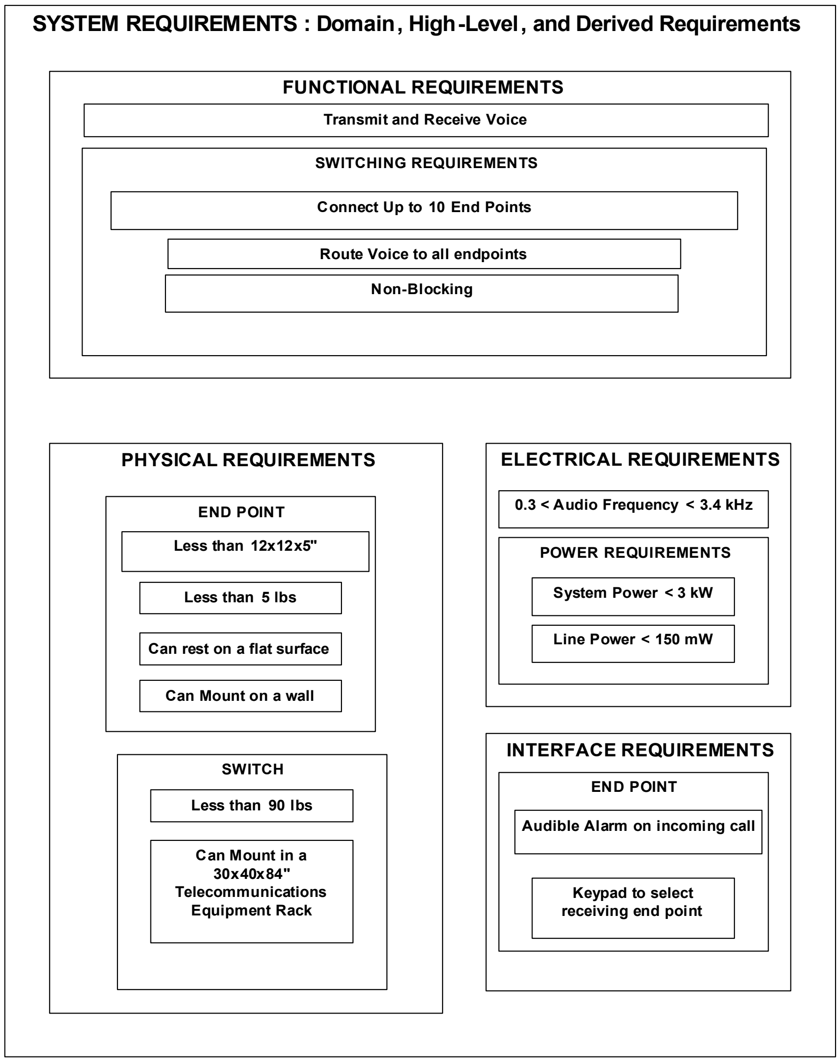

SYSTEM REQUIREMENTS : Domain, High-Level, and Derived Requirements
FUNCTIONAL REQUIREMENTS
Transmit and Receive Voice
SWITCHING REQUIREMENTS
Connect Up to 10 End Points
Route Voice to all endpoints
Non-Blocking
PHYSICAL REQUIREMENTS
END POINT
Less than 12x12x5"
Less than 5 lbs
Can rest on a flat surface
Can Mount on a wall
SWITCH
Less than 90 lbs
Can Mount in a 30x40x84" Telecommunications Equipment Rack
ELECTRICAL REQUIREMENTS
0.3 < Audio Frequency < 3.4 kHz
POWER REQUIREMENTS
System Power < 3 kW
Line Power < 150 mW
INTERFACE REQUIREMENTS
END POINT
Audible Alarm on incoming call
Keypad to select receiving end point

Figure 43: Higraph Model of Domain, High-Level, and Low-Level/Derived Requirements

Imagine trying to design a higraph to model an airplane or large software system using higraphs. Even if the massive amount of nodes and overlaps could be arranged to correctly show the desired information, it would almost be unusable by humans due to the massive amount of data shown at one time.

Harel, in [8], says of this issue:

> *"In practice, overlaps should probably be used somewhat sparingly, as overly overlapping blobs might detract from the clarity of the total diagram."*

Harel's point is certainly well taken, and is demonstrated briefly in the above example. However, in real-world applications that use system models, a massive amount of overlap is likely. Higraphs themselves do not solve visual complexities, nor will they be able to.

To conclude, higraphs are a useful tool for organizing and connecting data and information generated during the system engineering lifecycle. 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. Even for small problems, there is significant challenge in organizing the layout of information on a diagram in a way that maximizes "ease of interpretation" for the end user. Therefore, there must be a filter (or abstraction tool) that mines the higraph and presents only the desired information to the end-user. In other words, like UML and SysML, the higraph modeling language/formalism must be able to interface with a software tool to perform this filtering in order to be useful in industry.

Chapter 5: Formal Modeling of Higraphs

The purpose of this chapter is to show how Harel's mathematical formalism for higraphs can be adapted, and extended, to be useful in the systems engineering domain. This involves specifying ways to mathematically define the system requirements, structure, and behaviors models that make up the system model.

Higraph models of large systems (containing requirements, structure, and behavior models) will be incredibly detailed and complex. The visual representation will be unreadable and unusable by humans, unless computer software is developed to control the presentation of models to end users. Ideally, this software should present only the information/data that is relevant to a decision at hand, and should abstract all other information into the background. However, in order to develop software that will allow for the enterprise level use of the higraph formalism, there first needs to exist a mathematical and/or logical model that can be followed. To this end, and as summarized in Chapter 2, Harel has developed a valid mathematical and logical model for higraphs. What is left for us is to extend this mathematical and logical model to allow for the large scale system modeling that we propose higraphs be used for.

Section 5.1: Extended Mathematical and Logical Modeling of Higraphs

From Section 2.2 we know that a higraph can be completely defined (structurally) through the following higraph equation:

$H = (B, E, \rho, \Pi)$ where:

- B is the set of 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)

When this equation is applied to an actual system higraph, the result is a DAG for the system representation. From a systems engineering perspective, the formulation is missing a mapping of the other information shown in the system higraph (such as system component attributes and functions, system behaviors, etc.) on to the DAG. We therefore propose that equation 1 be extended to include assignment of types to nodes and edges in higraphs, and definitions to hierarchies and orthogonalities. More specifically, from the diagrams in Chapter 4 we see that nodes in Higraphs may represent (but are not limited to) the following:

- System Requirements
- System Structure
  - Component Attributes
  - Component Function
- System Behaviors

If B is the set of nodes 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_{2-1} and B_{2-2} (set of all system component attribute nodes, and set of all system component function nodes, respectively). So, $\mathbf{B = (B_1, B_2, B_3)}$ where $\mathbf{B_2 = (B_{2-1}, B_{2-2})}$

Similarly, higraph edges may represent (but are not limited to) the following:

- Assignment of Requirements
  - Assignment of a requirement to a component
  - Assignment of a requirement to a behavior
- Assignment of a behavior to a component (Functional allocation)
- Inheritance between system components
- A transition from one system state to another, corresponding to a behavior

If E is the set of edges that make up a Higraph, 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_{1-1} and E_{1-2} (set of all requirements assigned to system components, and set of all requirements assigned to system behaviors, respectively). So, $\mathbf{E = (E_1, E_2, E_3)}$ where $\mathbf{E_1 = (E_{1-1}, E_{1-2})}$

Hierarchy in Higraphs might represent (but is not limited to) the following:

- Derived Requirements
- System Component Specification
    - Allocation of an attributes to a component
    - Allocation of a function to a component
- 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, $\boldsymbol{\rho = (\rho_1, \rho_2, \rho_3)}$ where $\boldsymbol{\rho_2 = (\rho_{2\text{-}1}, \rho_{2\text{-}2})}$

Orthogonality in Higraphs may represent (but are not limited to) the following:

- Logical Partitioning of Requirements
    - Physical Requirements
    - Functional Requirements
- Structural Relationships
    - Composition Relationship in System Structure
    - Aggregation Relationship in System Structure
- Concurrent System Behaviors

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, $\boldsymbol{\Pi = (\Pi_1, \Pi_2, \Pi_3)}$ where $\boldsymbol{\Pi_1 = (\Pi_{1\text{-}1}, \Pi_{1\text{-}2})}$ and $\boldsymbol{\Pi_2 = (\Pi_{2\text{-}1}, \Pi_{2\text{-}2})}$.

Summary of Mathematical and Logical Higraph Definition

The above lists are not in any way exhaustive. Depending on the application, these definitions will vary widely, and can accommodate any user defined relationship. What we have defined in this section is summarized below in Table 3.

	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

Table 3: Summary of Higraph Element Definitions

Section 5.2: Software Representations of Higraph Models

Now, after showing the ability to define all connectivity (relationships) within a system model using higraphs, and providing a mathematical model to represent the overwhelming amount of visual information that will exist in even a small higraph based system model, the question remains, "What can we really do with this higraph model?" This is where the true strength of a higraph implementation of a system model lies—in the practical application

Following the higraph definition, and derived mathematical model, what we are left with is a large number of sets (sub-graphs, really) that define all of the relationships in a system. The formal definition of a higraph, shown in the equations that make up the higraph quadruple, could easily be modeled by computer software as a data structure, a relational (or entity based) database, or as metadata (an XML file, an Excel spreadsheet, etc.). As such, many of the uses described in the following section would be easy to accomplish given a mathematical definition, and software representation of the System Higraph.

More discussion on software implementation of a higraph modeling tool is discussed in Section 8.1.

Chapter 6: Practical Uses of Higraph Based System Models

Almost all applications of a complete system higraph model stem from a question (or query) asked of the model. The answer to these questions (or queries) will come from specific traces (or pathways) between higraph nodes along edges. Since our formal definition of the system higraph allows for user defined nodes, edges, hierarchies, and orthogonalities, and the higraph equation can be modeled in software, combining these two things allows for almost limitless traces and queries of the system higraph.

Section 6.1: Defining and Generating Custom Views of a System

Engineers, most of us anyway, do not build models of systems without having a use for that model. Many common uses are to develop complete and correct system designs, to provide some level of validation that a system will meet all requirements before it is implanted, and so on. In the case of higraphs, the traditional uses of models can be satisfied, as well as some additional, very practical uses.

Most real-world systems contain too much data and information to work with simultaneously at any one time. To keep the complexity of systems engineering processes in check when dealing with these real-world systems, customized viewpoints are created—the viewpoint shows only that information necessary to communicate the required data used to make a decision in the systems engineering process.

One viewpoint may show all requirements associated with a system component (for the vendor who will need to provide that component), or another viewpoint may show all prices associated with a specific subsystem (for the manager responsible for producing that subsystem). Using a higraph model of a system, generating such a viewpoint is only a matter of following a select group of edges from specific nodes in the higraph.

If you want to find all requirements associated with a specific system component, you only need to trace all "requirements" edges coming from the component node in the higraph. More likely, a software program would trace through the representation of the higraph following rules defined by the user for edge types, and node types, beginning with the node (component) in question, and tracing all edges (requirements) connected to it.

Likewise, if you want to find all costs associated with a specific subsystem, your need to pull all cost attributes from the components that make up this subsystem. Again, it is most likely a software program would generate this data based on user definitions of the nodes (subsystem, subsystem components, component cost attributes).

Section 6.2: Evaluating the Impact of Requirements Change

One of the most useful applications that a complete higraph model of a system would allow is the ability to determine what system components and behaviors are affected by a requirements change. Requirements change often throughout the course of engineering projects, and while existing software applications (DOORS, Rational) allow for the maintenance of system requirements, what is really needed is to find out exactly what impact changing a requirement has on the system in terms of system structure and system behavior.

Since the higraph model would connect all requirements to system components and behaviors, and would allocate all behaviors to a system component, tracing from a new/modified requirement to all of the affected components (attributes and behaviors) would be straightforward.

For instance, if the total amount of available power for a system component was modified, a trace from this requirement to the power attribute of the component in question would present the engineer with information about whether the new requirement can be satisfied with the existing component.

At a higher level, say the total amount of power available for the entire system was modified, a trace through the Higraph would follow all applicable levels of derived requirements and system structure hierarchy to identify and “roll-up” all power specifications for the system (as currently designed). Again, this information would be presented to the engineer responsible for designing and validating system power consumption.

Section 6.3: System Validation

By virtue of the many types of edges allowed in the higraph (requirement assignment, allocation of behavior, complies with, satisfies, etc.), tracing these edges will reveal much information about the validity of the system design. For instance edge inspection will ensure:

- 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.
- All system behavior nodes (system behaviors/functions) are can be traced to a system structure node (system component). This ensures correct functional allocation; all behaviors are allocated to a specific component function.
- No system structure or behavior nodes exist that can not be traced to a requirement. This would imply extraneous components or behaviors that are not needed (aka. “Gold Plating”)
- The system structure is an instance of the domain structure (for normal, non-innovative, systems). This ensures that what you will build is in line with existing principals. Likewise, ensure system behaviors comply with domain behaviors (again, for normal, non-innovative systems). This ensures that all behaviors follow existing principals (physical laws).

Note, these sorts of validations do not take the place of system testing. While system testing ensures the system was built correctly, system validation ensures the correct system was built.

Chapter 7: Higraph Enabled Modeling of an Existing System

This chapter presents a detailed example of how higraphs can be used as a system modeling tool. This example focuses on the modeling of a typical office network, consisting of workstations for each user, servers for email and file storage, and so forth. To simplify the model development, we will assume that the network is already in place—therefore, requirements are known and have been defined, all components are selected and are in place, and so forth. Thus, the principal goal is to demonstrate how the system can be represented in higraph form, and that the higraph model can be used to respond to queries and changes to system requirements.

A similar exercise could take place for a system design problem, where the initial design of the system (from CONOP, to architecture, to requirements, structure, and behavior models) has yet to take place.

The scope of this example includes presentation of system requirements (including Use Cases that define system behaviors), presentation of large pieces of the system structure, presentation of large pieces of the system behavior, and documentation of the relationships among system requirements, structure, and behavior. The example will also develop pieces of the math model from which we will perform pseudo-queries of the system higraph model to show how helpful information can be gained.

Section 7.1: System Requirements Model

Table 4 contains the complete list of system requirements to which the office network complies with. As noted, for this example (to show how a specific point on higraph representation of Use Cases) the behavior requirements are all specified in the Use Case diagrams that specify this system.

Requirement #	Requirement Area	Requirement Type	Requirement	Requirement Owner
1	System	Explicit	The system shall provide an office network for running work related applications	Corporate
1.1	Structure	Explicit	The system shall use a client/server architecture	IT
1.1.1	Structure-Apps	Explicit	The system shall support software applications	Engineering
1.1.1.1	Structure-Apps	Explicit	The system shall support email communications	Engineering
1.1.1.1.1	Structure-Apps-Email	Derived	The system shall support SMTP email	IT
1.1.1.1.2	Structure-Apps-Email	Derived	The system shall support POP3 email	IT
1.1.1.2	Structure-Apps	Explicit	The system shall support word processing	Engineering
1.1.1.3	Structure-Apps	Explicit	The system shall support voice over IP (VoIP) communications	Engineering
1.1.1.4	Structure-Apps	Explicit	The system shall support printing	Engineering
1.1.1.4.1	Structure-Apps-Print	Explicit	The system shall provide one printer for every 50 users	IT
1.1.1.5	Structure-Apps	Explicit	The system shall support internet browsing	Engineering
1.1.2	Structure-Network	Explicit	The system shall operate on a TCP/IP network	IT
1.1.2.1	Structure-Network	Explicit	The system shall operate at 100Mbps on the LAN	IT
1.1.2.1.1	Structure-Network-LAN	Derived	The system shall use CAT5 network connections on the LAN	IT
1.1.2.1.1.1	Structure-Network-LAN	Derived	The system components shall be collocated within 100m	IT
1.1.2.2	Structure-Network	Explicit	The system shall interface with a T3 WAN link	IT
1.1.3	Structure-Storage	Explicit	The system shall provide storage for user data	IT
1.1.3.1	Structure-Storage	Explicit	The system shall provide 20GB of storage per user	IT
1.1.3.2	Structure-Storage	Explicit	The system shall provide 5TB of common storage	IT
1.1.4	Structure-Security	Explicit	The system shall allow security features	Security
1.1.4.1	Structure-Security	Explicit	The access to each drive shall be controlled	Security
1.1.4.1.1	Structure-Security-Drives	Explicit	The system shall allow individual user access to user drives	Security
1.1.4.1.2	Structure-Security-Drives	Explicit	The system shall allow total access to common drives	Security
1.1.4.2	Structure-Security	Explicit	The system shall provide backup capabilities	Security
1.1.4.2.1	Structure-Security-Backup	Explicit	The system chall provide configuration backup capabilities	Security
1.1.4.2.2	Structure-Security-Backup	Explicit	The system shall provide file backup capabilities	Security
1.1.4.3	Structure-Security	Explicit	The system shall require unique usernames and passwords for access	Security
1.1.4.4	Structure-Security	Explicit	The system shall allow administrator audits of all user activities	Security
1.1.5	Structure-Users	Explicit	The system shall allow 100 users	Corporate
1.2	Cost	Explicit	The system shall be cost effective	Finance
1.2.1	Cost	Explicit	The system shall cost less than $300,000 in initial setup costs	Finance
1.2.2	Cost	Explicit	The system shall be made from commercially available items	Finance
1.3	Power	Explicit	The system shall consume less than 17,000 Watts	Facilities
1.3.1	Power	Explicit	The system shall run on 110V power input	Facilities
1.3.1.1	Power	Derived	The system shall consume less than 154 Amps	Facilities
1.4	Behavioral	Explicit	SEE USE CASES	Engineering

Table 4: Complete List of Office Network System Requirements

All requirements in this system have the following information associated with them: Unique requirement number (the structure of which dictates a requirement hierarchy), requirement area (structure, behavioral, cost, power, etc.), requirement type (explicit or derived), requirement owner (corporate, finance, engineering, IT, security), and finally the requirement text. As a higraph, an individual requirement node would look like this:

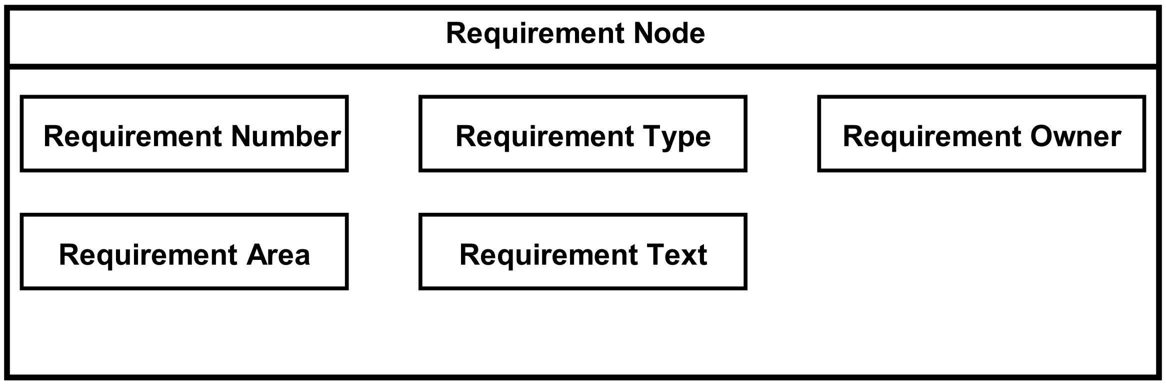


Figure 44: Higraph Requirement Node

Modeled as a higraph node with all information shown, Requirement 1.2.1 from Table 4 would look like:

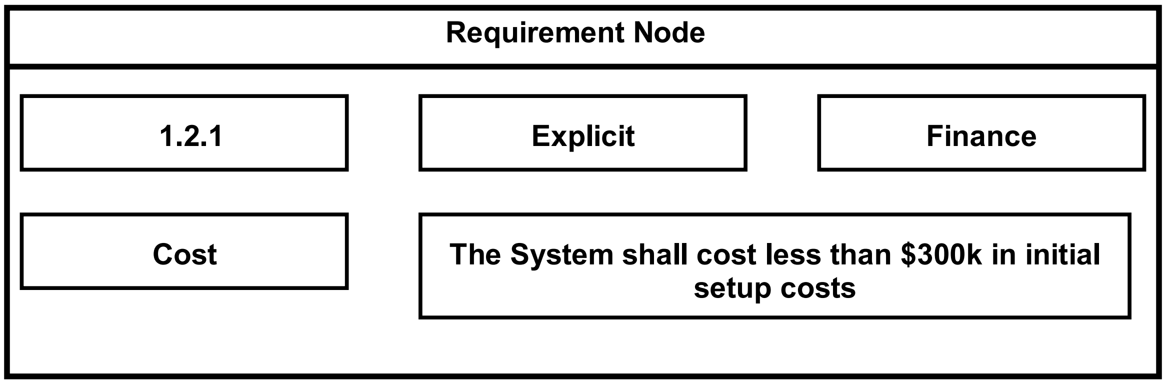

Requirement Node
1.2.1
Explicit
Finance
Cost
The System shall cost less than $300k in initial setup costs

Figure 45: Requirement Node for Requirement 1.2.1

Many of the higraphs used in this example will not show all information in all views. Indeed, most will show only the requirement text. It is important to note that the information does exist, and is contained in the mathematical equation that makes up the higraph.

Organization of Requirements

To simplify the interpretation of requirements and assignment of requirements to project developers (or other project teams), requirements are organized into areas corresponding to those defined in Table 4 (column 2). Working from the high level requirements down, a higraph representation of requirements would consist of the following diagrams:

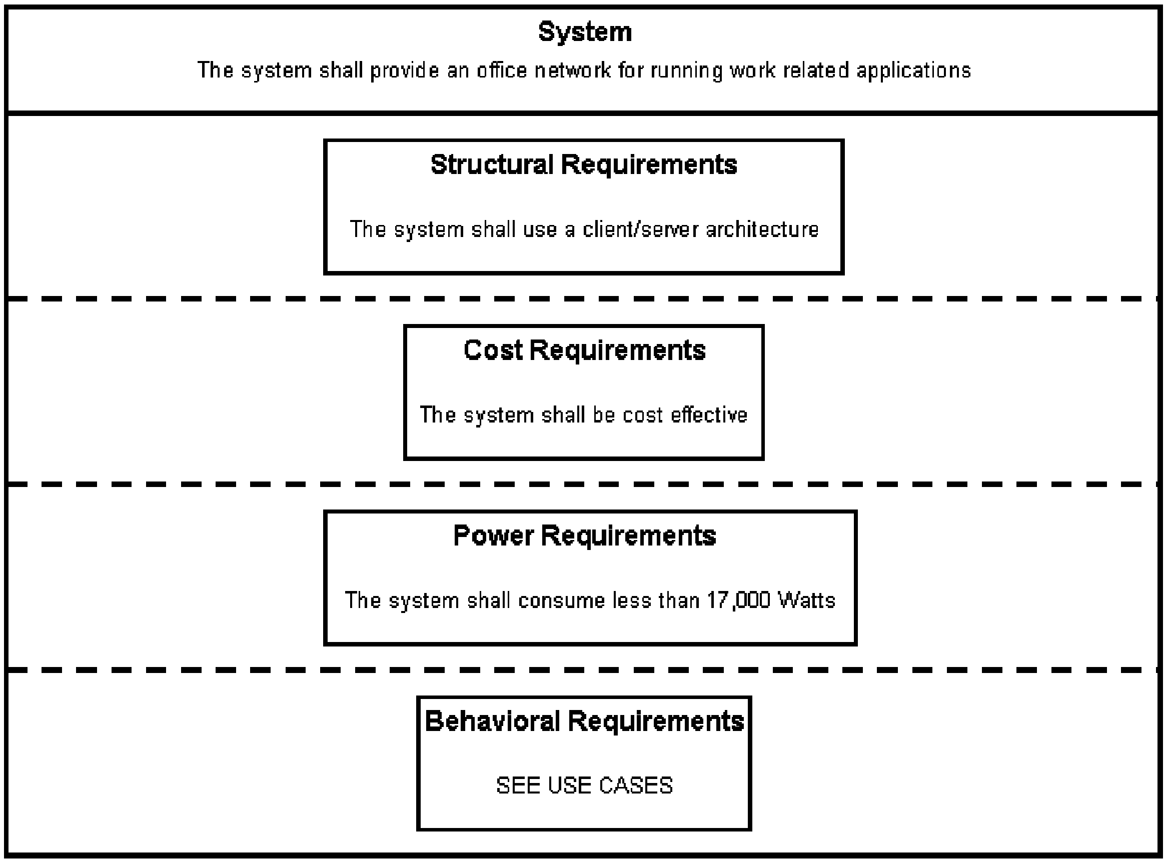

System
The system shall provide an office network for running work related applications
Structural Requirements
The system shall use a client/server architecture
Cost Requirements
The system shall be cost effective
Power Requirements
The system shall consume less than 17,000 Watts
Behavioral Requirements
SEE USE CASES

Figure 46: System Requirements Higraph: Top Level

At a high level, the system will have four types of requirements (derived from each of the four orthogonal regions). There will be structural requirements, cost requirements, power requirements, and behavior requirements. As we will show, the behavior requirements are captured in a Use Case diagram (or a Use Case higraph) as noted in the higraph above.

The structural requirements are shown in the higraph below (representing a lower level in the requirements hierarchy)

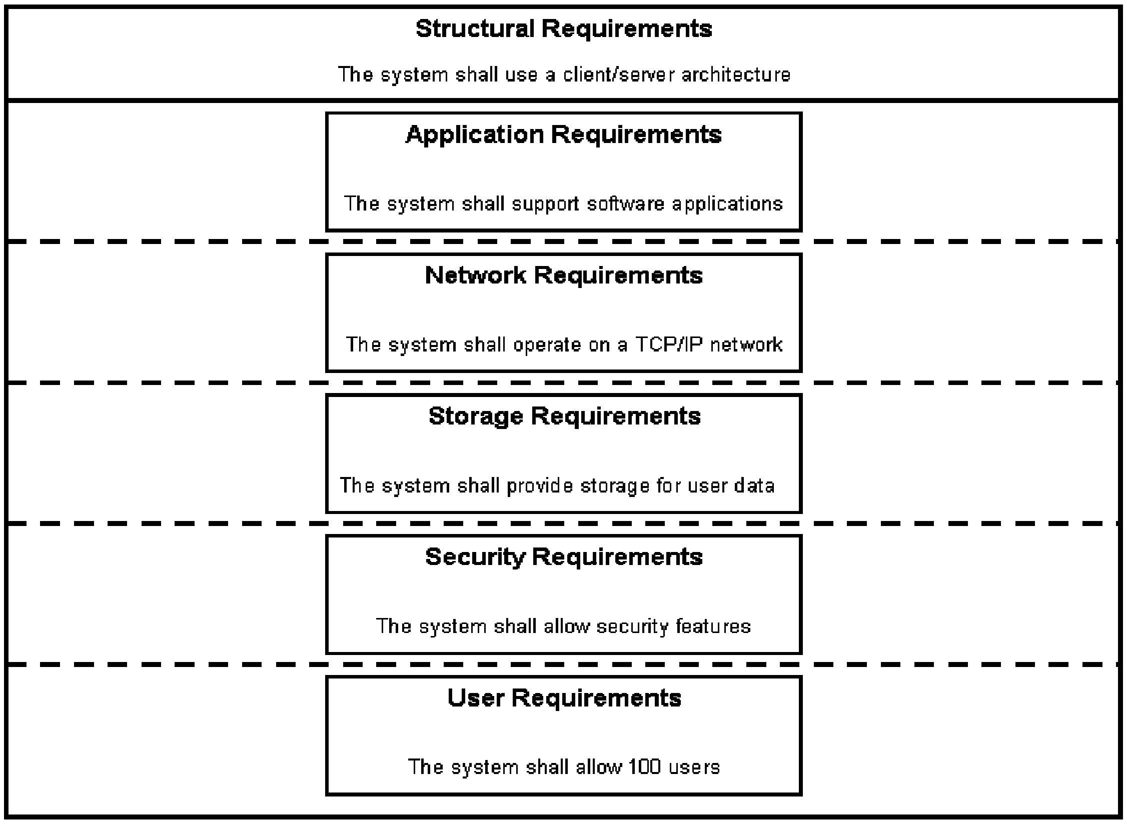

Structural Requirements
The system shall use a client/server architecture
Application Requirements
The system shall support software applications
Network Requirements
The system shall operate on a TCP/IP network
Storage Requirements
The system shall provide storage for user data
Security Requirements
The system shall allow security features
User Requirements
The system shall allow 100 users

Figure 47: System Requirements Higraph: Structural Requirements

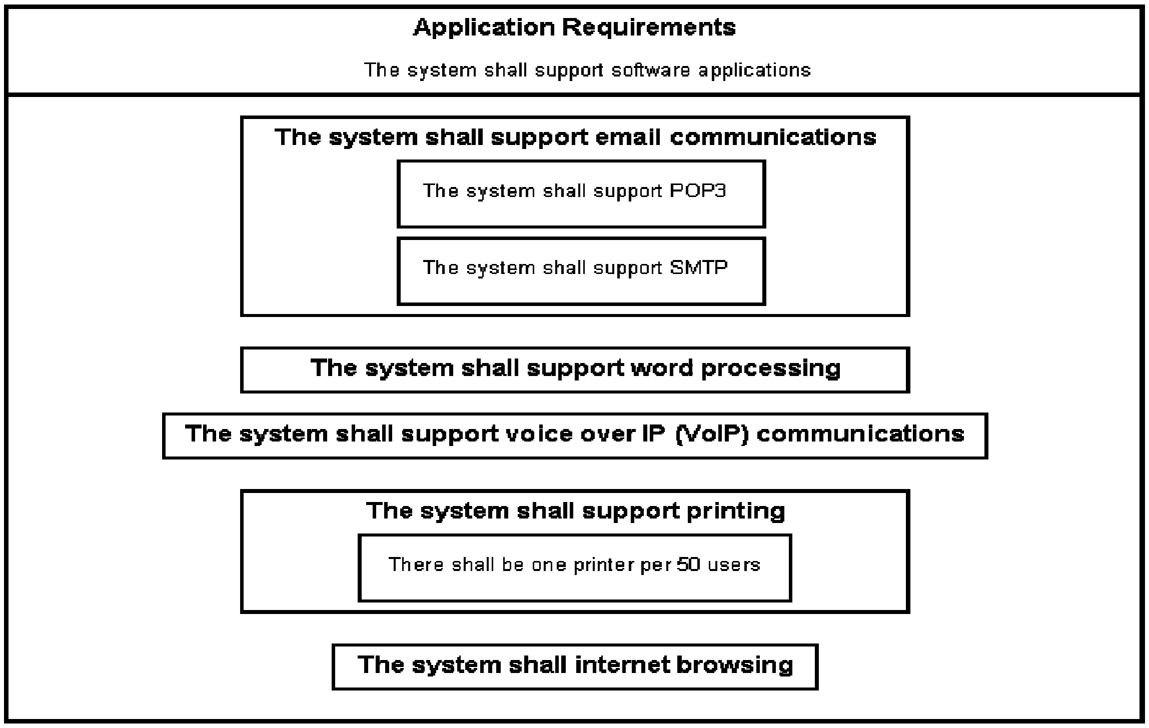

Application Requirements
The system shall support software applications
The system shall support email communications
The system shall support POP3
The system shall support SMTP
The system shall support word processing
The system shall support voice over IP (VoIP) communications
The system shall support printing
There shall be one printer per 50 users
The system shall internet browsing

Figure 48: System Requirements Higraph: Application Requirements

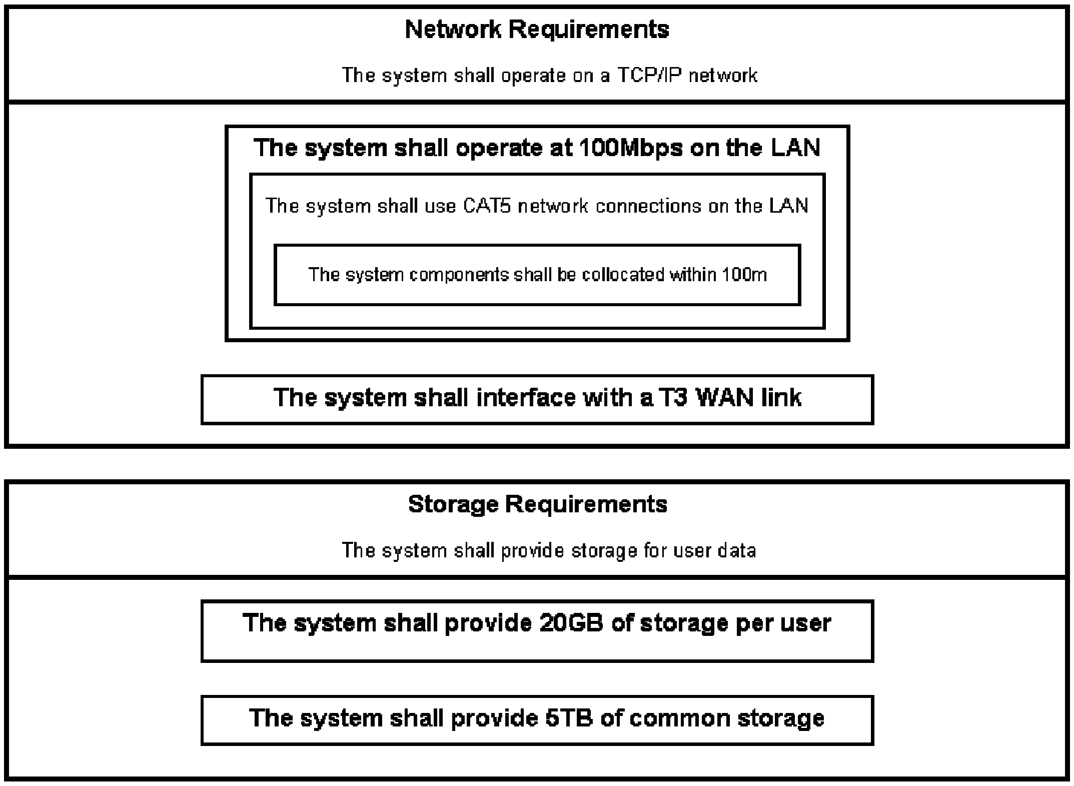

Network Requirements
The system shall operate on a TCP/IP network
The system shall operate at 100Mbps on the LAN
The system shall use CAT5 network connections on the LAN
The system components shall be collocated within 100m
The system shall interface with a T3 WAN link
Storage Requirements
The system shall provide storage for user data
The system shall provide 20GB of storage per user
The system shall provide 5TB of common storage

Figure 49: System Requirements Higraph: Network and Storage Requirements

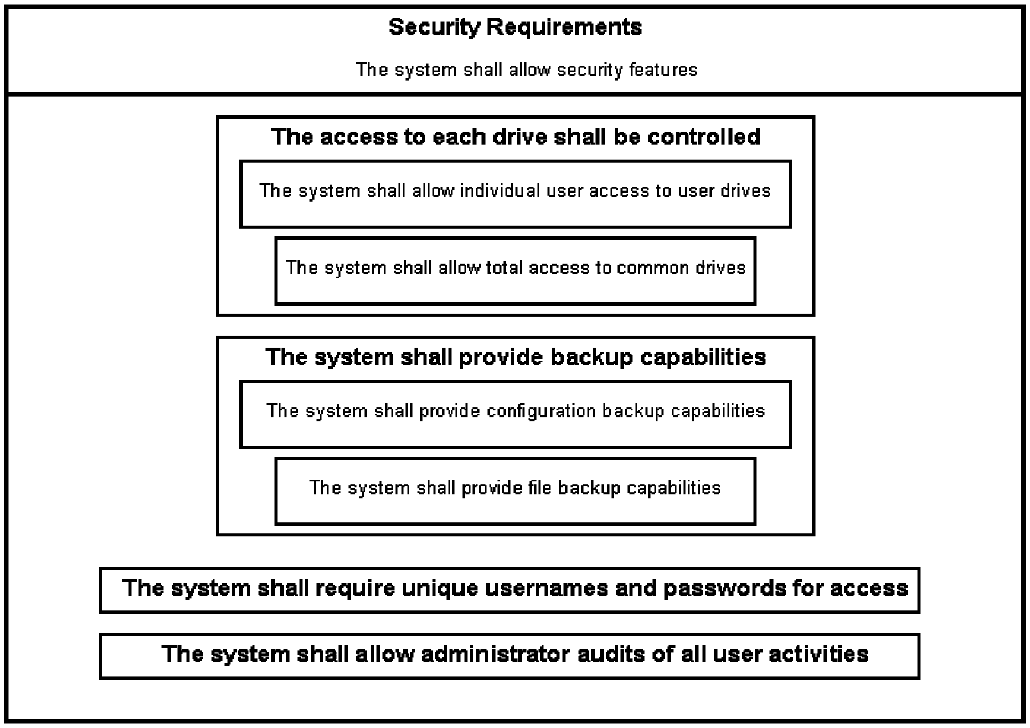

Security Requirements
The system shall allow security features
The access to each drive shall be controlled
The system shall allow individual user access to user drives
The system shall allow total access to common drives
The system shall provide backup capabilities
The system shall provide configuration backup capabilities
The system shall provide file backup capabilities
The system shall require unique usernames and passwords for access
The system shall allow administrator audits of all user activities

Figure 50: System Requirements Higraph: Security Requirements

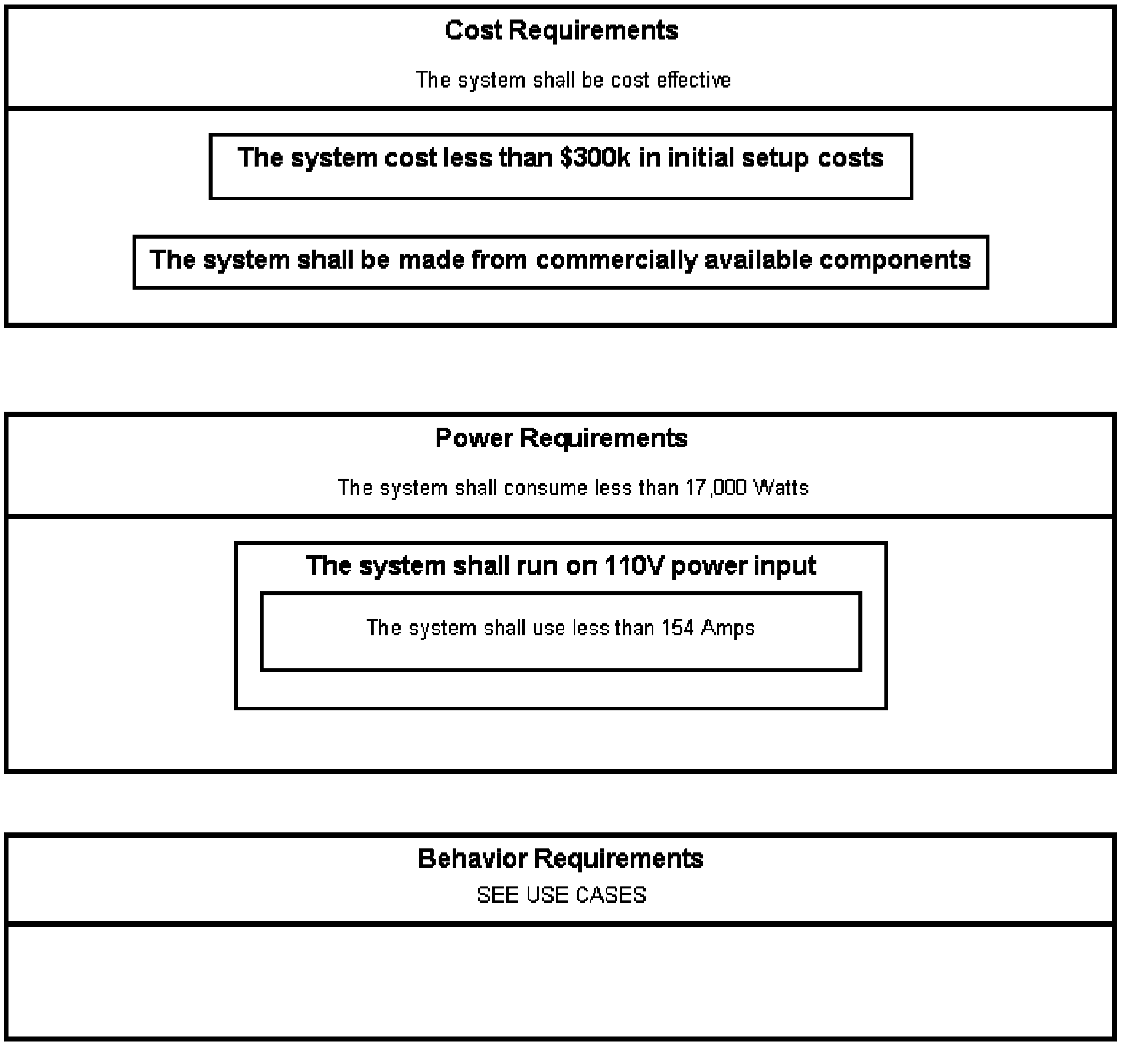

Cost Requirements
The system shall be cost effective
The system cost less than $300k in initial setup costs
The system shall be made from commercially available components
Power Requirements
The system shall consume less than 17,000 Watts
The system shall run on 110V power input
The system shall use less than 154 Amps
Behavior Requirements
SEE USE CASES

Figure 51: System Requirements Higraph: Cost, Power, Behavior Requirements

Additionally, a separate set of domain requirements are shown in Figure 52 below. The requirements represent existing rules that relate to a system in this (network) domain. These requirements should be satisfied by this system's design unless otherwise noted.

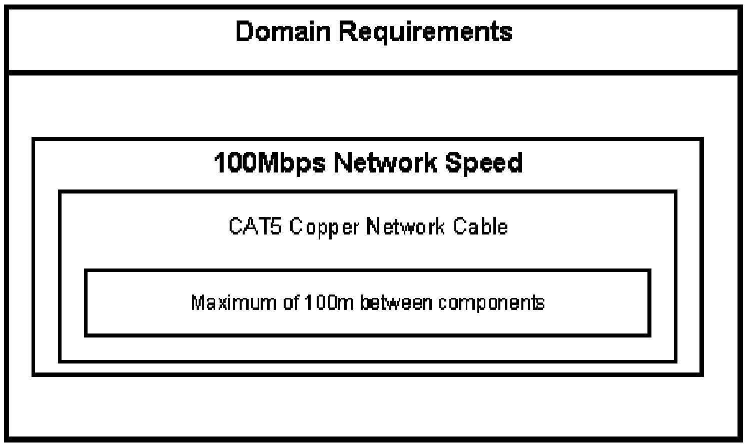


Figure 52: System Requirements Higraph: Domain Requirements

Behavior Requirements

As noted in Figure 51, the required system functionality is documented in a higraph representation of a UML/SysML Use Case diagram.

The top-level functionality is shown in Figure 53. The relationship between the actors and lower-level tasks are shown in Figures 54 through 56. By organizing the required system functionality into a hierarchy of higraphs, each level of presentation is considerably simpler than if we attempt to show all aspects of the functionality in a single diagram.

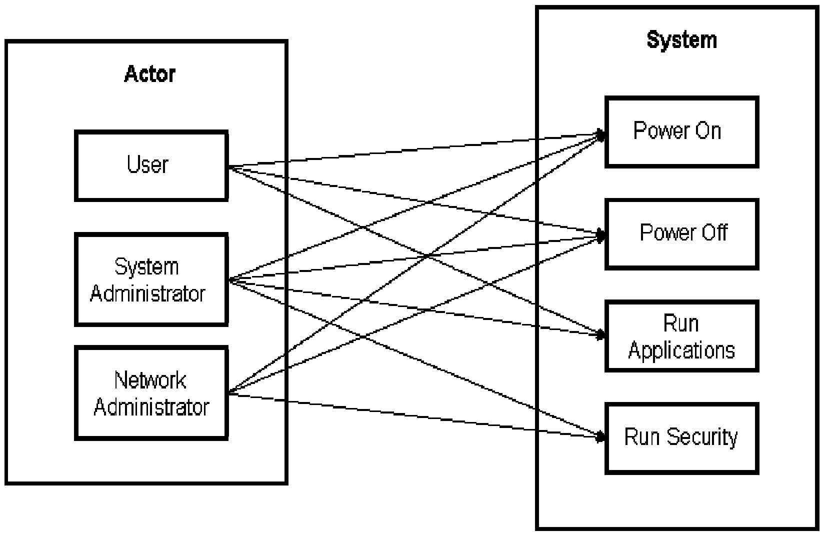


Figure 53: System Use Case Higraph: Top Level System Functionality

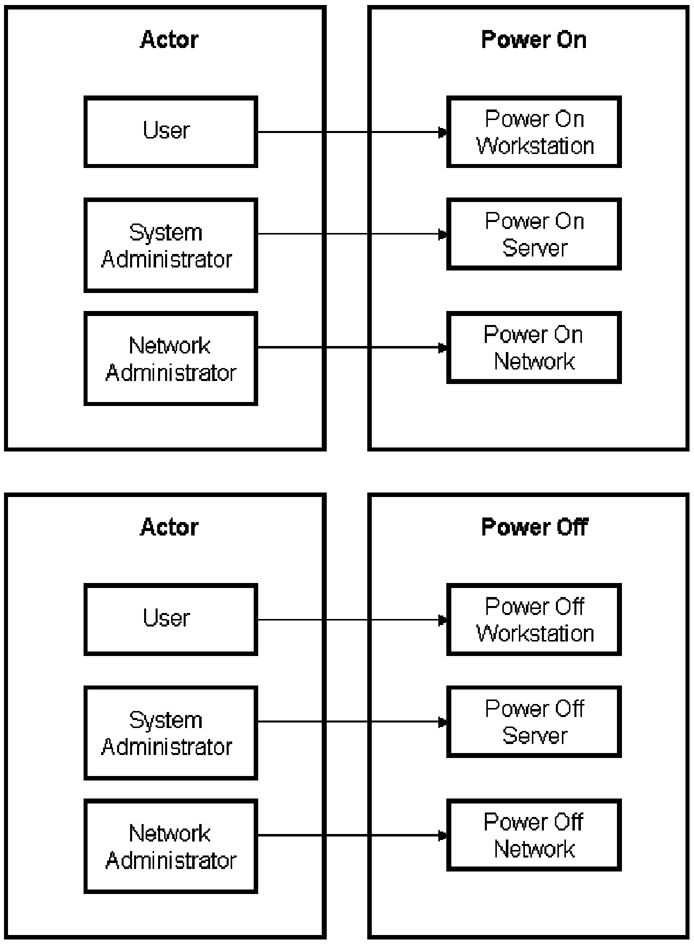


Figure 54: System Use Case Higraph: Power On/Off

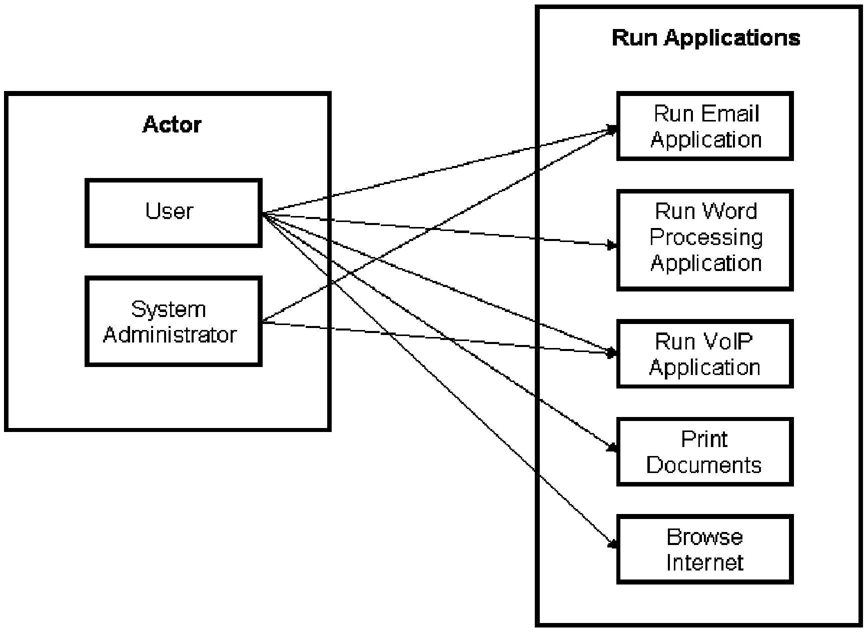


Figure 55: System Use Case Higraph: Run Applications

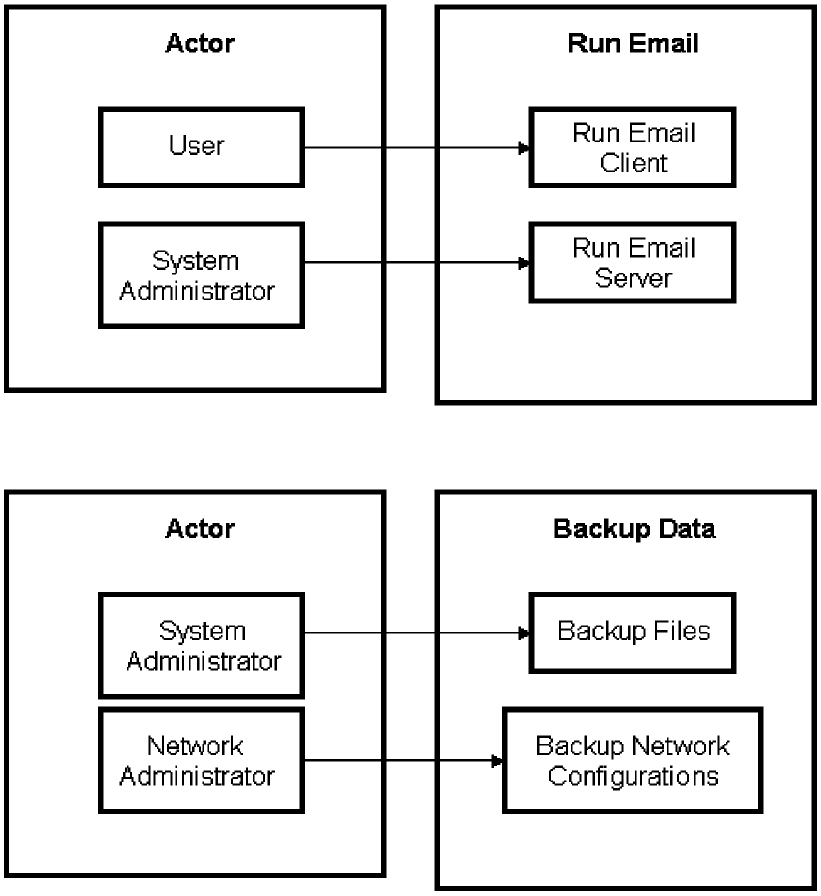


Figure 56: System Use Case Higraph: Run Email, Backup Data

Section 7.2 System Structure Model

Now that we have documented the existing system requirements in higraph form, we will move on to modeling system structure. In this system, we consider two types of system components: hardware and software. The highest level system structure higraph is shown below.

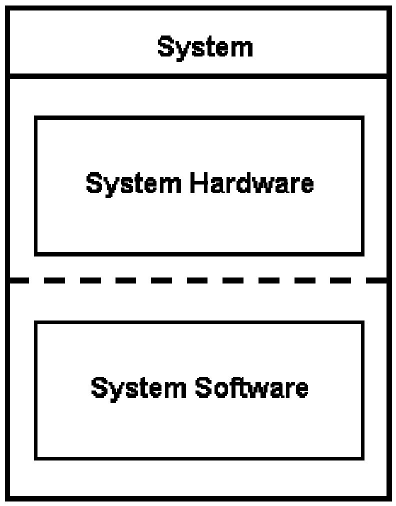


Figure 57: System Structure Higraph: Top Level

Hardware and software are placed in orthogonal regions since they are fundamentally different types of components.

Hardware and software components (classes) are made up of attributes and functions as shown in the figures below.

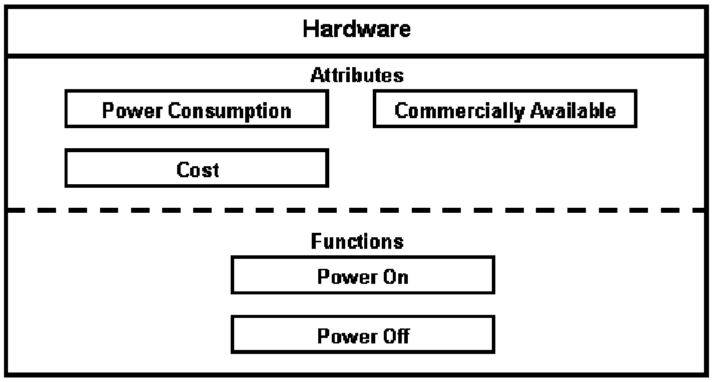


Figure 58: System Structure Higraph: Hardware

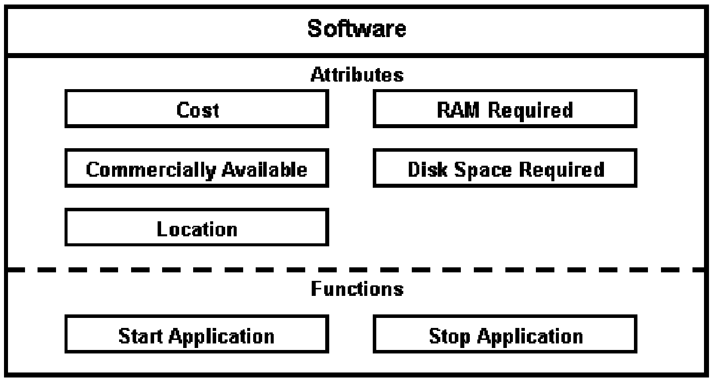


Figure 59: System Structure Higraph: Software

The hardware class is inherited by different types of hardware shown in the higraph below.

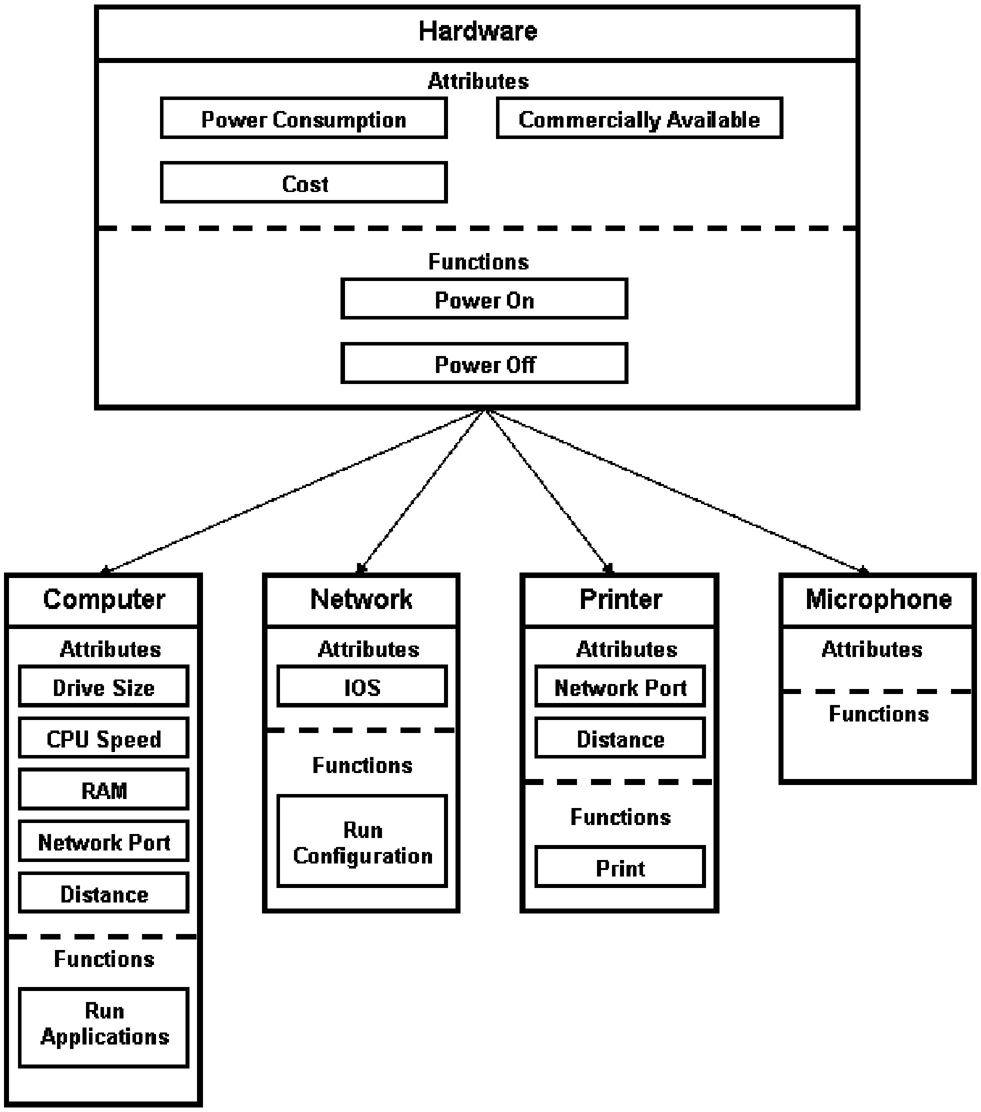


Figure 60: System Structure Higraph: Hardware Inheritance

The above higraph shows that the computer, network, printer, and microphone nodes (classes) all include the attributes and functions shown in the hardware node. Thus, higraphs are used to show inheritance in an object oriented manner.

The network class can be broken down further as in the higraph below.

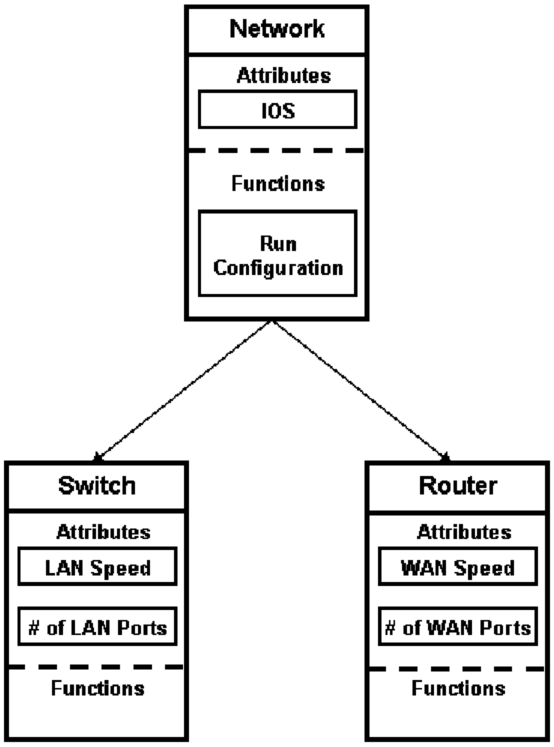


Figure 61: System Structure Higraph: Network Inheritance

So, the Switch and Router nodes note only inherit the attributes and functions from their direct parent, the Network node, but also from the Hardware node through the inheritance shown by the directed edges in this higraph.

The higraph representing the software structure is shown below. The same principals of inheritance apply as a result of directed edges between the nodes.

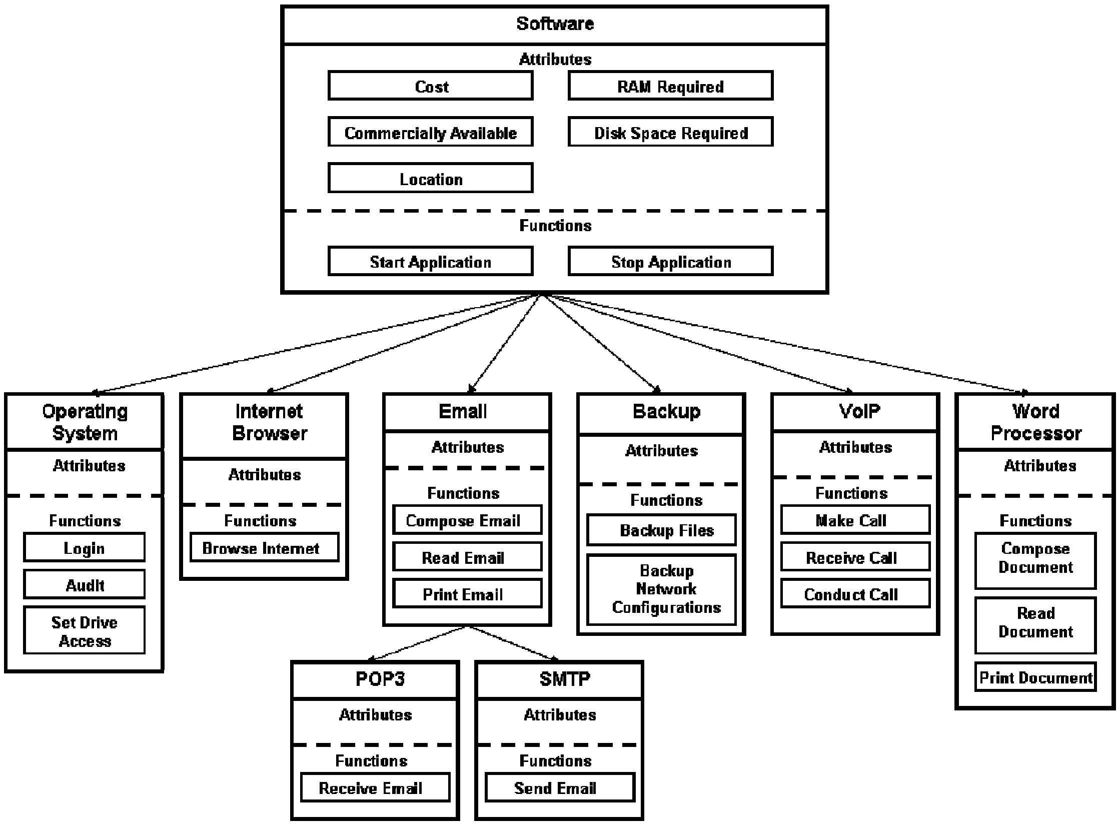


Figure 62: System Structure Higraph: Software Inheritance

Implementation View of System Structure

To complete the system structure model, we need to create an implementation view showing the specific hardware and software components used in this system. Again, this is represented by a higraph. Edges in this case show multiplicities, that is the number of a given component in relation to another component in the system. The progression of figures below shows the types and number of hardware and software components that will make up this system.

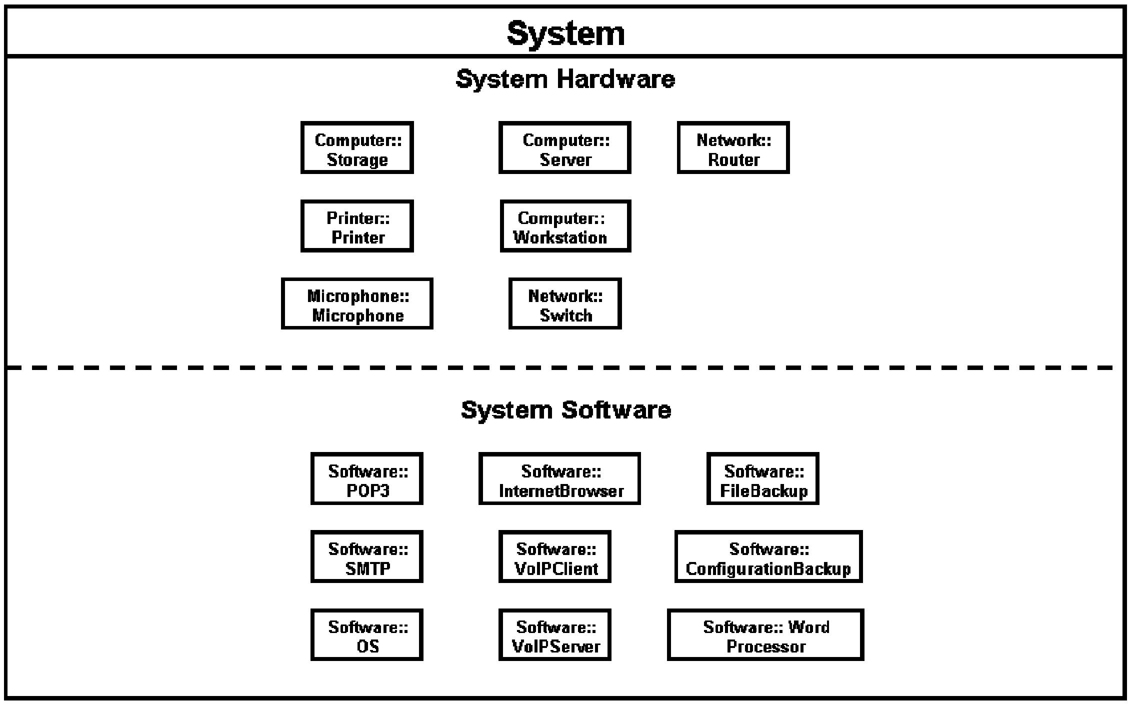


Figure 63: System Structure Higraph: Hardware and Software Implementation

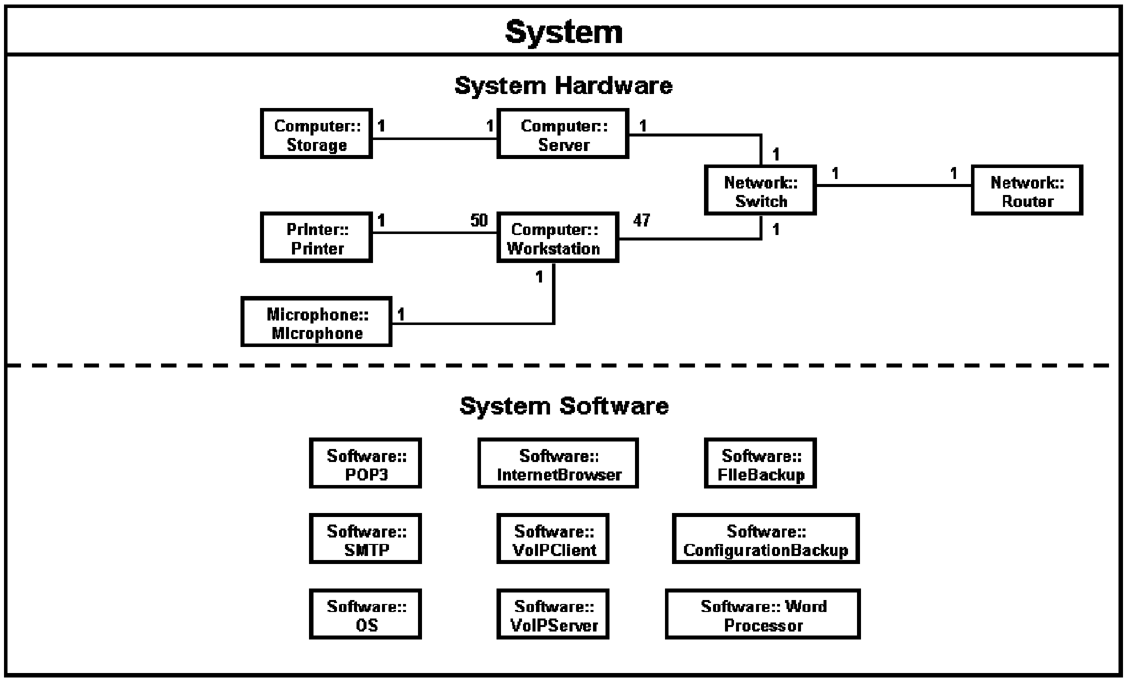


Figure 64: System Structure Higraph: Hardware Multiplicity

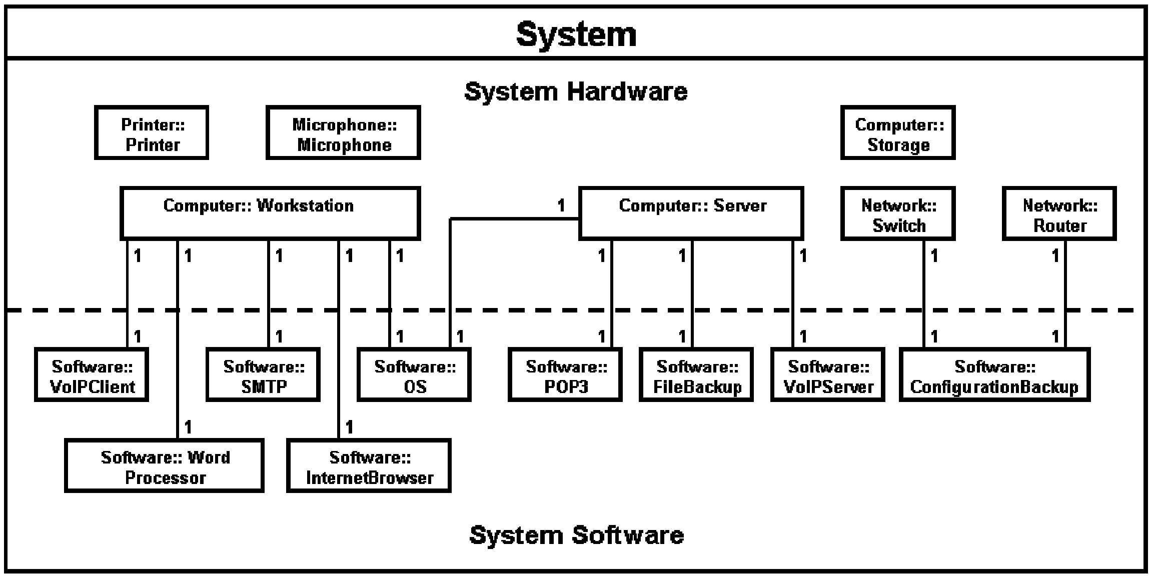


Figure 65: System Structure Higraph: Software Multiplicity

Section 7.3: System Behavior Model

Like requirements and structure, the system behaviors are represented in higraphs. In the behavior higraphs, nodes are system states, edges are system functions that cause a transition from one state to another, and orthogonality represents allowed concurrent behaviors.

The highest level of system behavior is modeled in the following higraph.

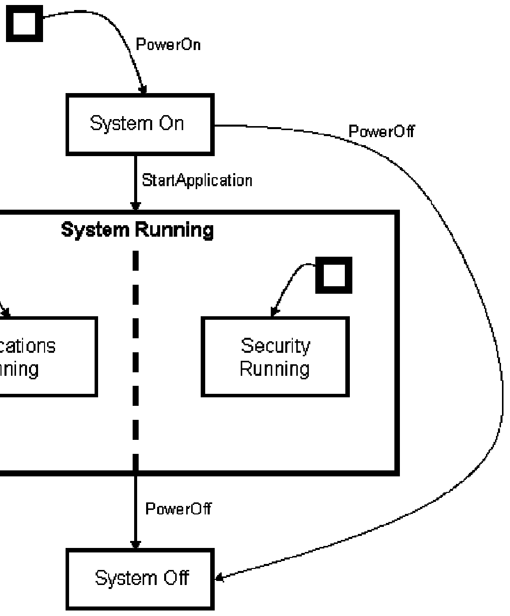


Figure 66: System Behavior Higraph: Top Level

This higraph shows that the allowed system states are On, Running, and Off. The system can be turned off at any point, represented by edges from all other states to the System Off state. Inside of the Running state, concurrent behaviors for applications running and security running are shown.

The following three diagrams show in more detail the behaviors in the System On state, and the transition to the system running state. What is important to note is the way the edges between the two states are drawn. As background, referring to the use cases, for applications to be running, the entire system (workstation, server, network) must be on. For security features to be running, only the server and network must be on. The third diagram below, Figure 69, shows the optimal way for edges to be drawn, but all three are correct higraphs.

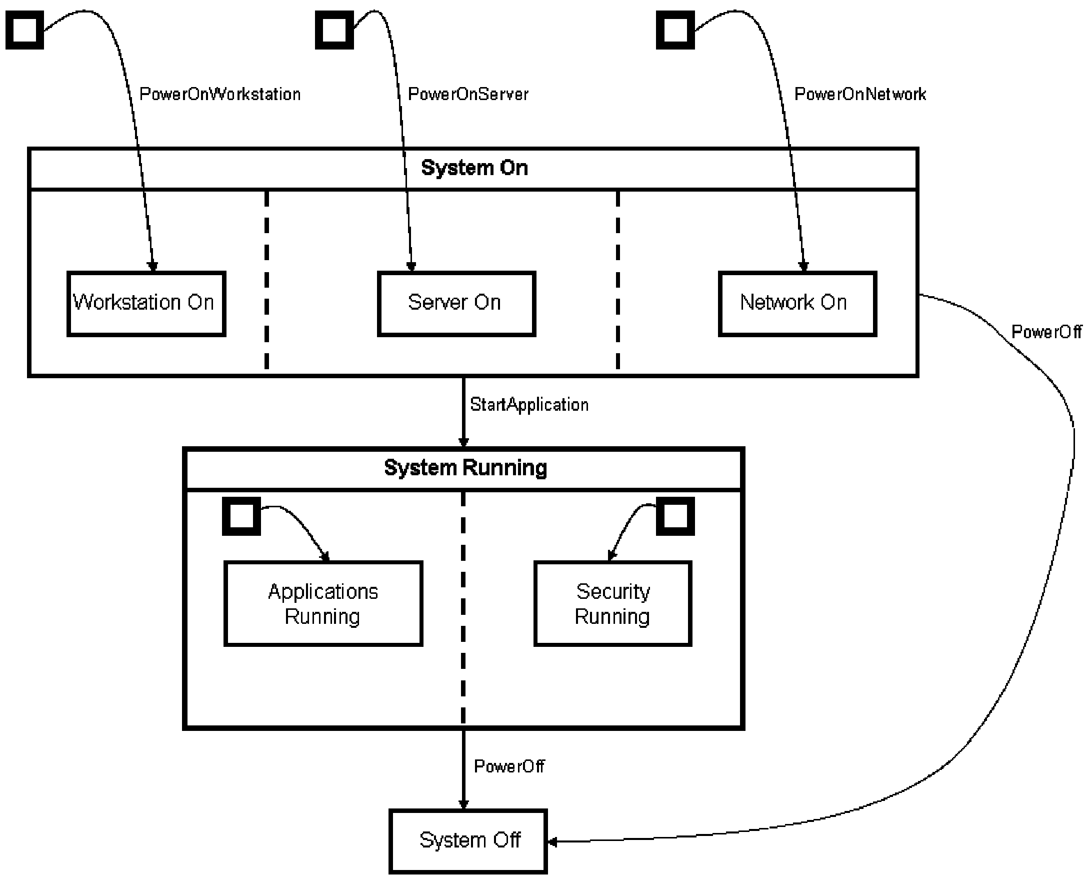


Figure 67: System Behavior Higraph: Minimal Edges

Without details shown, the edge from System On to System Running indicated that some, or all, of the behaviors in System On must be completed before some, or all, of the behaviors in System Running can occur.

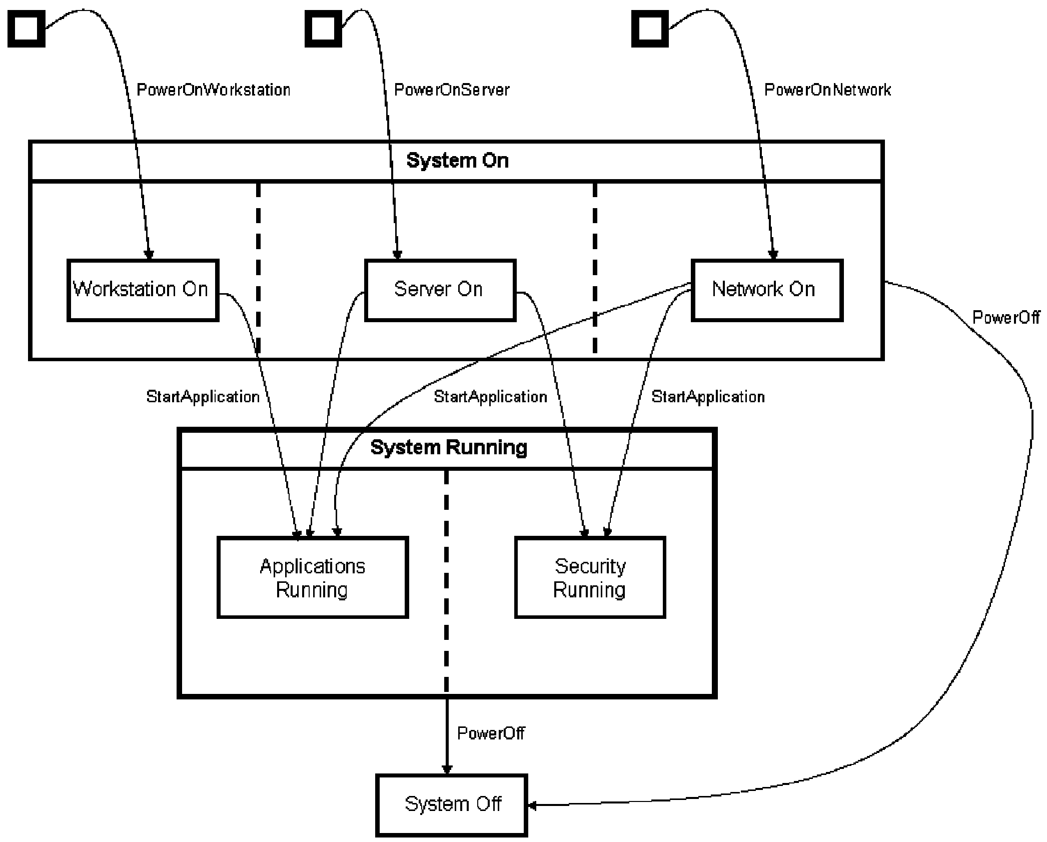


Figure 68: System Behavior Higraph: Maximum Edges

Here now we see that the workstation, server, and network must all be on before applications can run, **or**, the server and network must be on before security can run. However, a better way to show this, minimizing the edges, is shown in the next higraph.

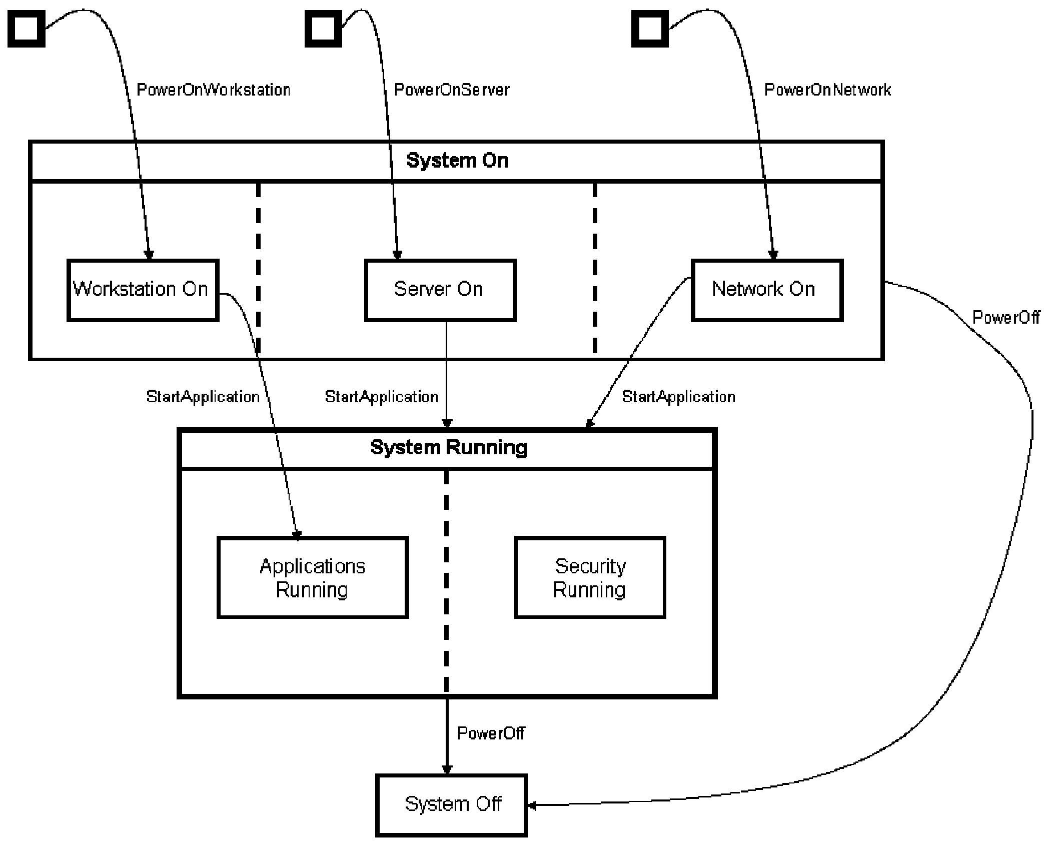


Figure 69: System Behavior Higraph: Optimal Edges

Since the server on and network on behaviors must occur before any of the behaviors in system running can occur, edges are shown to the system running node. The workstation need only be on for applications to run, so this edge is shown from the workstation on state to the applications running state directly.

Like all other higraphs, behavior higraphs use hierarchies to represent different levels of behavior. The next two higraphs show decreasing levels of hierarchy in the Applications Running state.

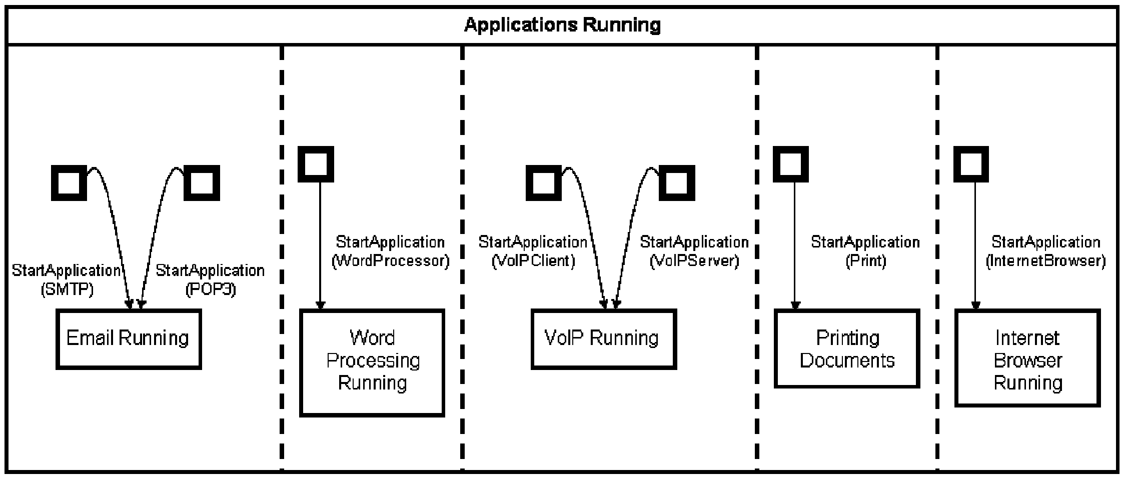
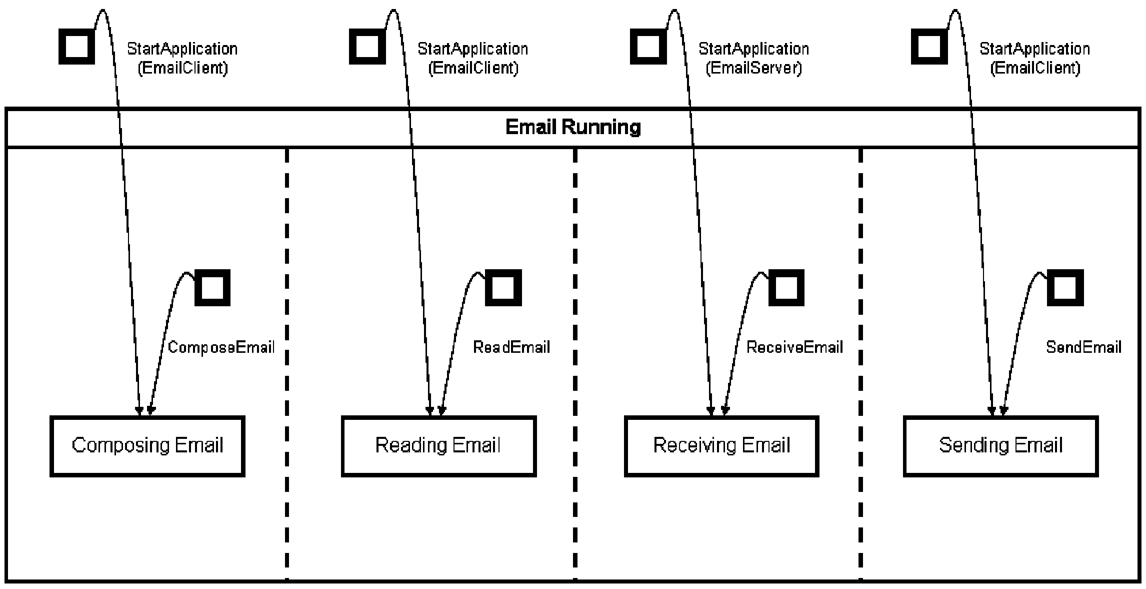


Figure 70: System Behavior Higraph: Applications Running



StartApplication (EmailClient)
StartApplication (EmailClient)
StartApplication (EmailServer)
StartApplication (EmailClient)
Email Running
ComposeEmail
ReadEmail
ReceiveEmail
SendEmail
Composing Email
Reading Email
Receiving Email
Sending Email

Figure 71: System Behavior Higraph: Email Running

Section 7.4: Structural Requirements Traceability

To this point we have presented higraphs that represent a substantial portion of system requirements, system structure, and system behavior. Now we will show examples of connectivity, via edges, between system components. These edges will represent the concepts discussed in Section 4.4 such as allocation of requirements to component attributes and system behaviors, allocation of system behaviors to component functions, and traces from domain requirements to system requirements.

The figure below shows the allocation of the system cost requirement to attributes in system structure components—system hardware components and system software components in this case.

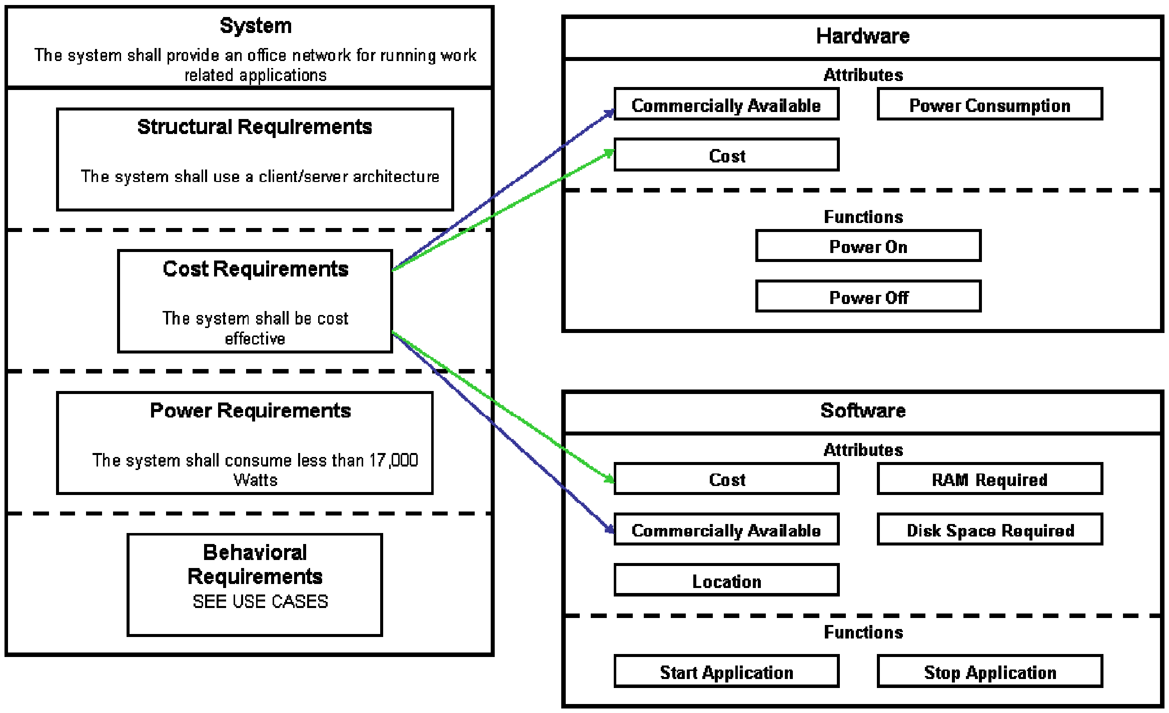


Figure 72: System Higraph Model: Cost Requirements Allocation I

Looking lower in the hierarchy of system cost requirements, we get Figure 73 below. **Remember, these figures do not exist on their own, but rather they are created as the result of a query of the system higraph model.** In these cases, the query would ask “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.

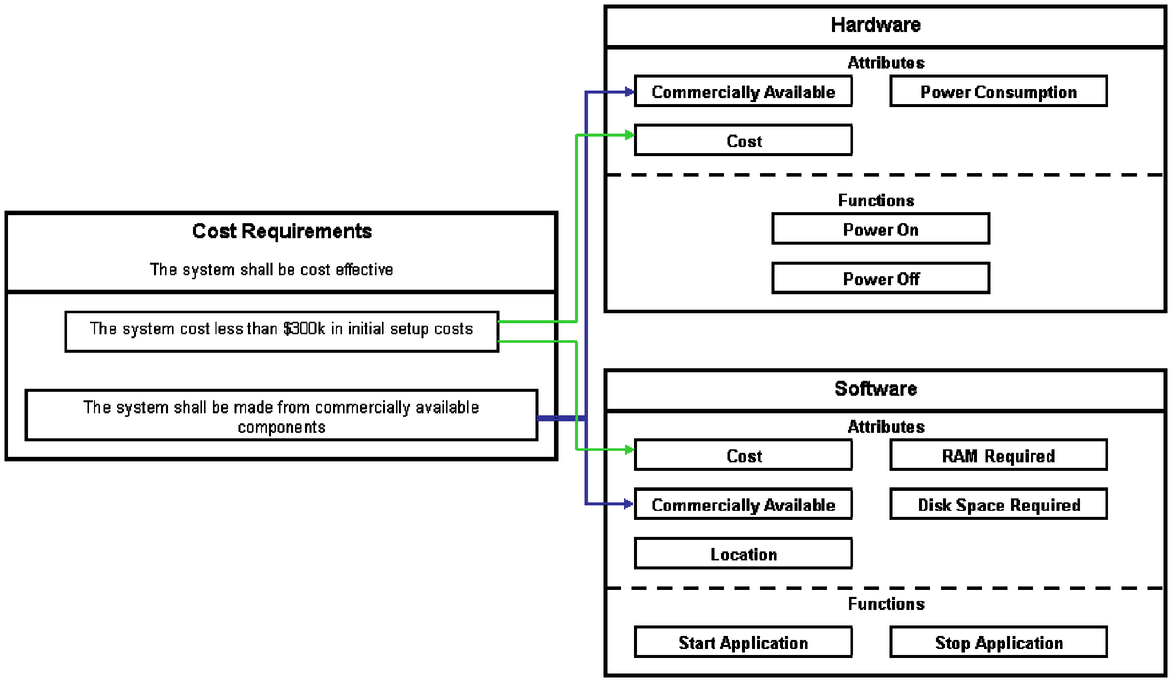


Figure 73: System Higraph Model: Cost Requirements Allocation II

A similar example is shown below for the power requirements. Figure 73 would be produced as a result of a query for “Show all system attributes that satisfy the system power requirements.” Here, we find out that only hardware components are needed to satisfy system power requirements. This of course intuitively makes sense as software (used in the basic office applications required by this example) does not affect system power levels.

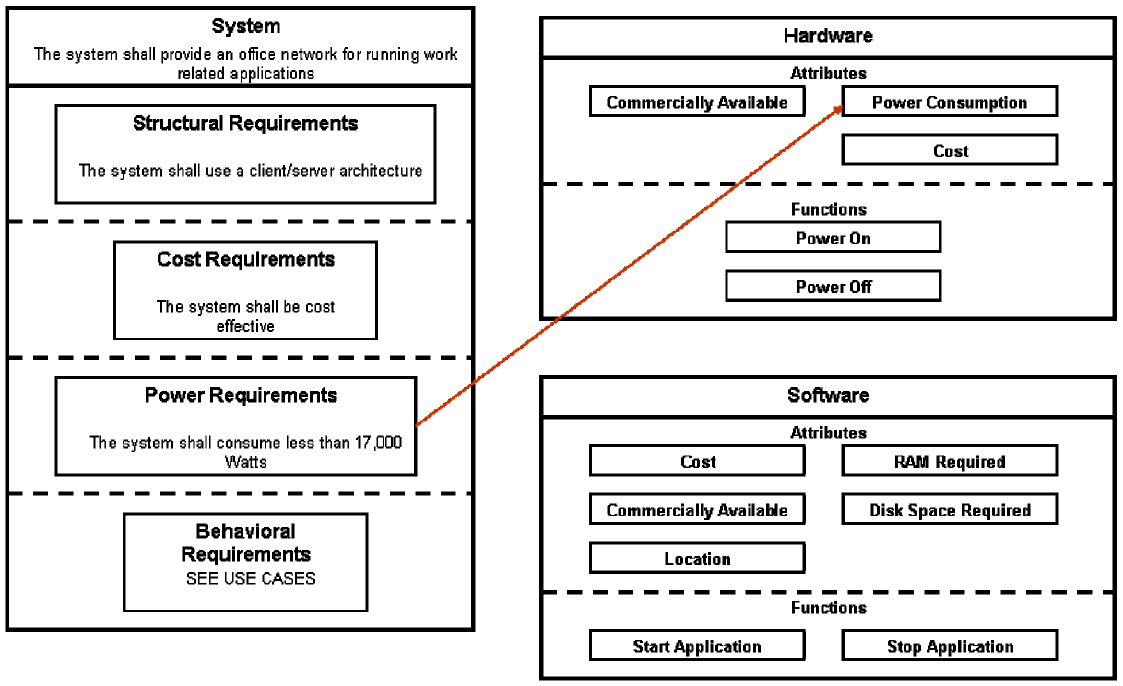
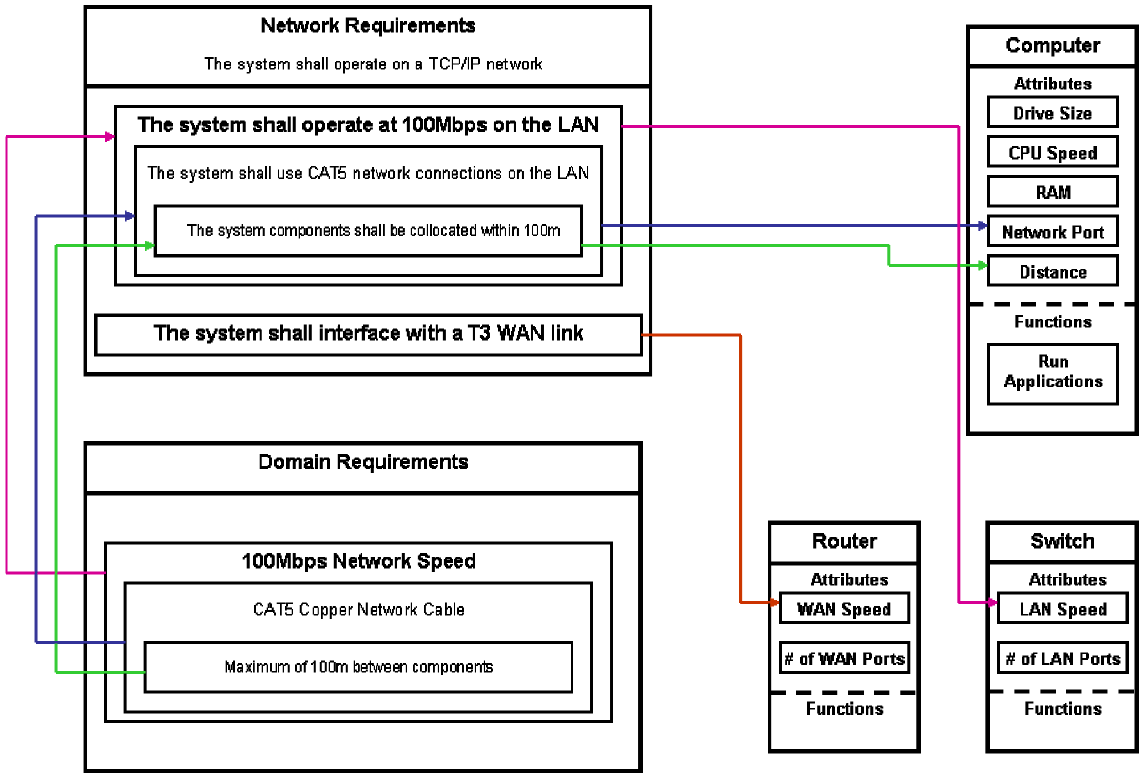


Figure 74: System Higraph Model: Power Requirements Allocation

Recalling the concept discussed in Section 4.4, domain requirements must be satisfied by system requirements for a system to work. In this example, the domain requirements deal with the physical limitations of a network operating at 100Mbps. 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 100Mbps, the domain requirements become applicable, and must trace to system requirements. The figure below shows this trace, as well as the allocation of these system requirements to system component attributes.



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

Figure 75: System Higraph Model: Domain Requirements Allocation

Again, the above figure does not necessarily exist independently from the system higraph. Instead, this figure would be created as a result of some query of the system higraph model. In this case the query might ask "Show how domain requirements trace to system requirements, and how those system requirements are allocated to system components." However, we see there is a connection from a network requirement (not specified by any domain requirements) to the Router component's WAN speed attribute. So, the true query for Figure 75 above would be "Show all relationships with system network requirements."

The last example in requirements traceability is shown in Figure 76 below. In this case, the higraph shows the association between the system's behavior requirements and the system use case higraph.

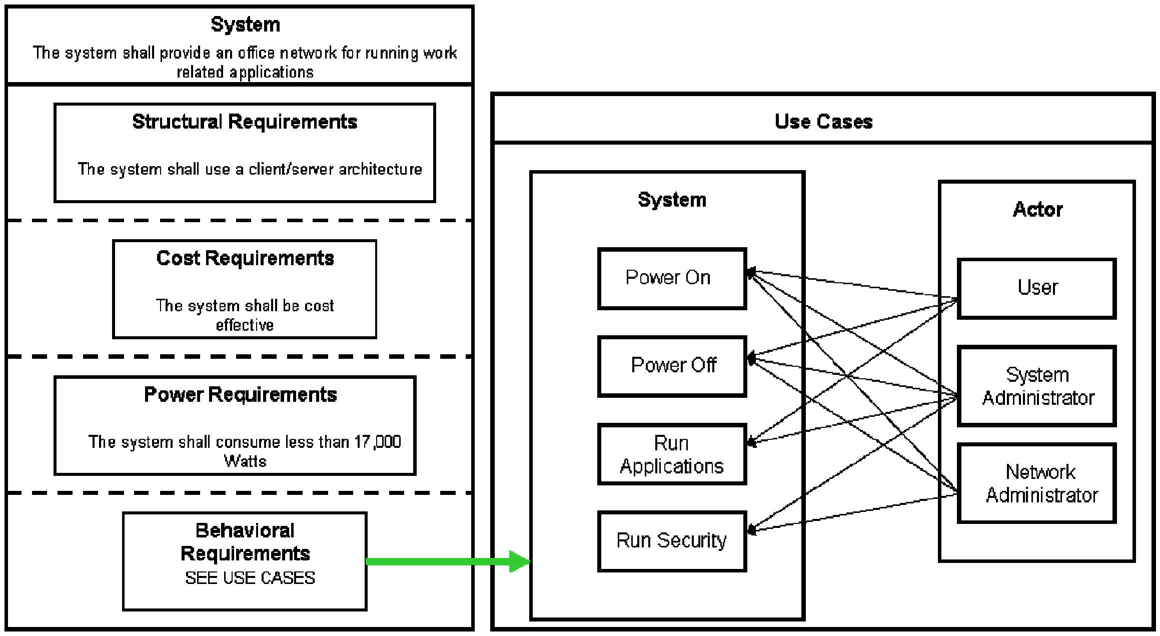

System
The system shall provide an office network for running work related applications
Structural Requirements
The system shall use a client/server architecture
Cost Requirements
The system shall be cost effective
Power Requirements
The system shall consume less than 17,000 Watts
Behavioral Requirements
SEE USE CASES
Use Cases
System
Power On
Power Off
Run Applications
Run Security
Actor
User
System Administrator
Network Administrator

Figure 76: System Higraph Model: Behavior Requirements Association

Section 7.5: Behavioral Requirements Traceability

In this section we have focused primarily on requirements traceability to system structure. As noted before, the same principles hold true for traceability to system behaviors. The figure below shows how the high level behavior requirements captured in the system use case higraph (see Figure 53) are allocated to system states (see Figure 66).

As usual, the higraph shown in the figure below does not exist outside of the complete system higraph model. Rather, this higraph would be derived from a query of the system model asking “What states satisfy system behavior requirements?”

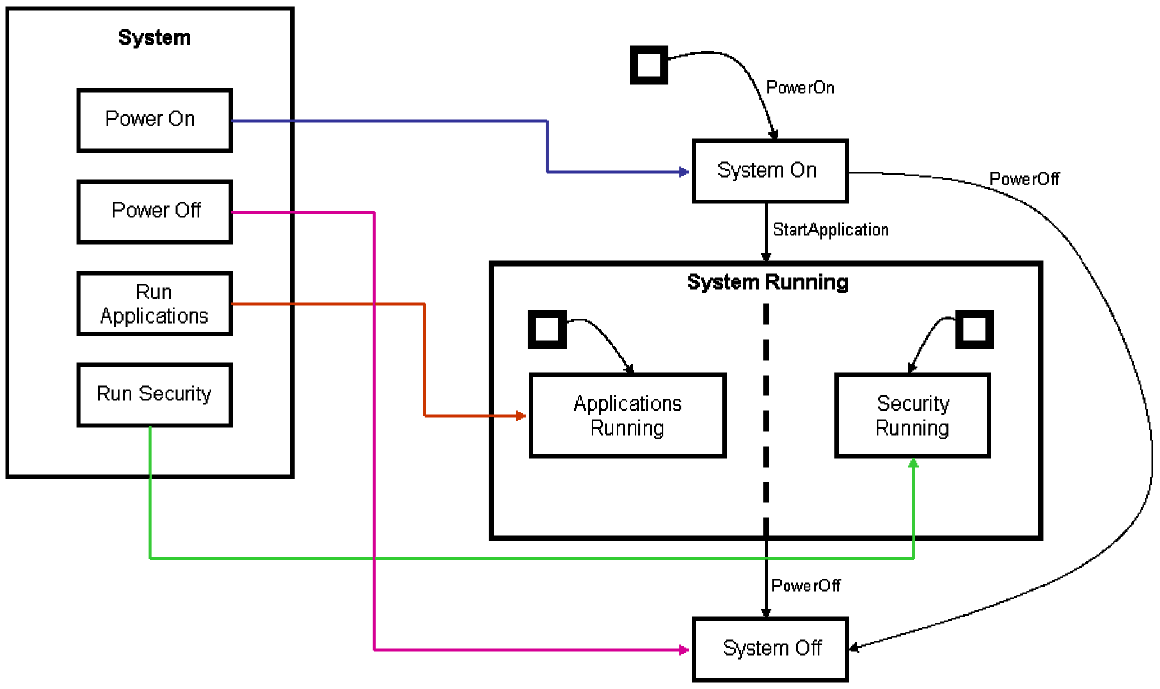


Figure 77: System Higraph Model: Behavior Traceability I

The higraph shown in Figure 78 below, also dealing with traceability of system behavior requirements (from the use case diagram in Figure 55) to system states (from the behavior diagram in Figure 70), would be generated from a query of the system higraph model asking “Show all system states that satisfy the behavior requirements for the Run Applications use case.”

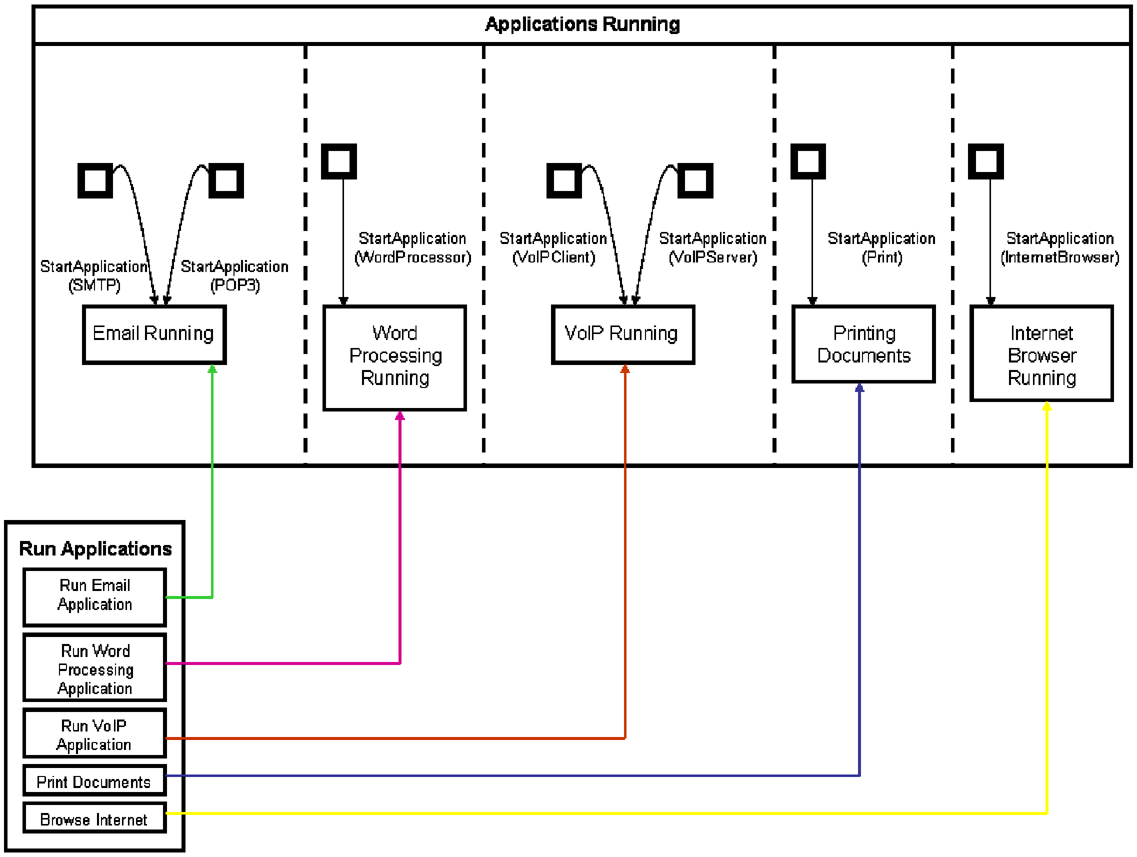


Figure 78: System Higraph Model: Behavior Traceability II

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.

The higraph in Figure 79 below shows an example of this. 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. In this case the functions are allocated to the email software, POP3 software, and SMTP software components.

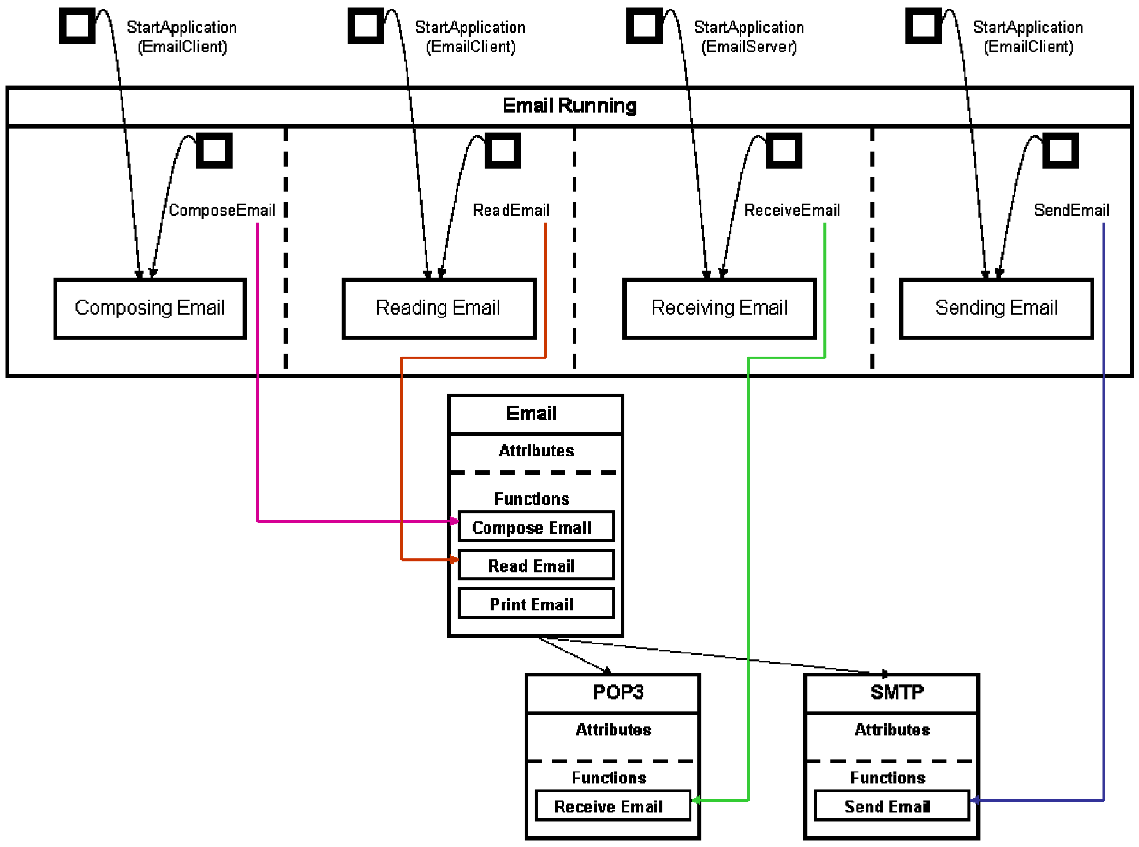


Figure 79: System Higraph Model: Behavior Allocation I

It is important to note the direction of the colored edges above. 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 email node to the POP3 and SMTP nodes imply inheritance (not allocation). This would be specified, as outlined in Section 5.1, through a user's definition of edges used in the system higraph model.

There is still some information missing from Figure 79 above. The Compose, Read, Receive, and SendEmail behaviors are allocated to system component functions, but the StartApplication() behaviors remain unallocated. Again, it is important to note that just because those allocations are not shown in a specific higraph diagram (say Figure 79 above) these allocations still should exist. A new query of the system higraph model could produce the higraph shown below in Figure 80.

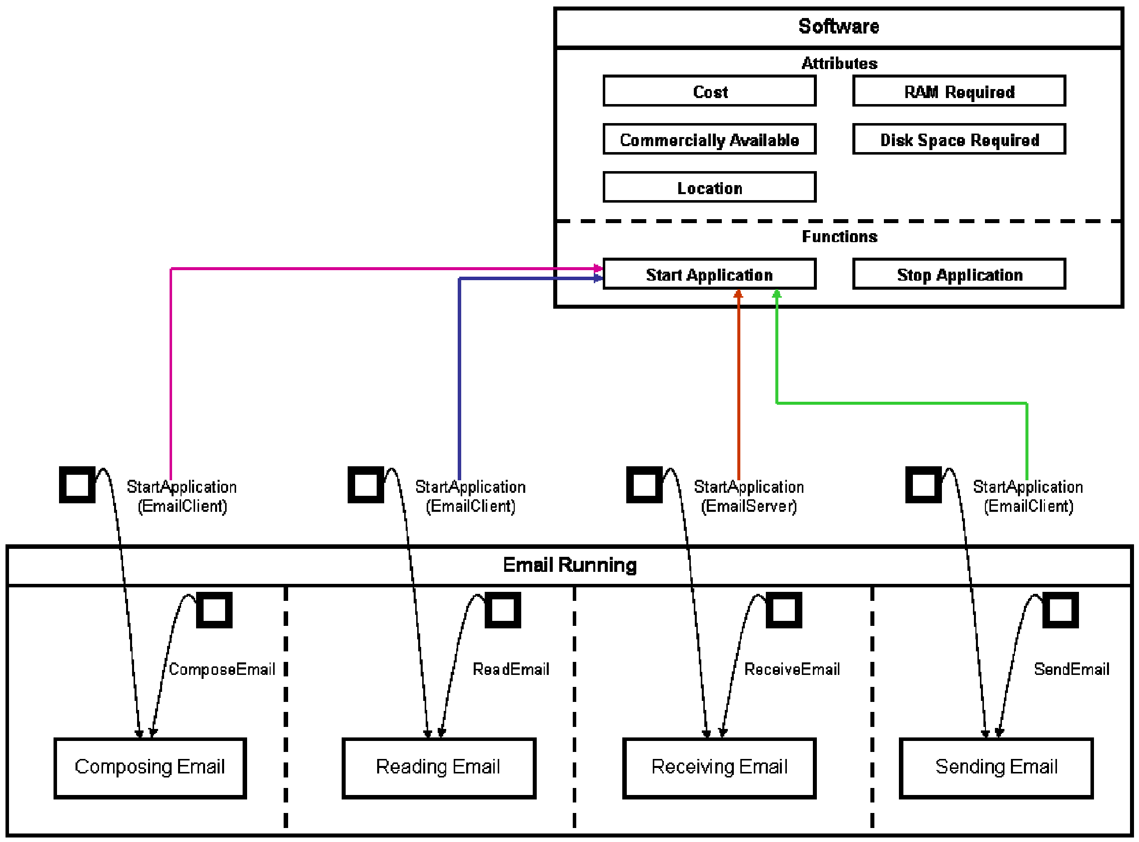


Figure 80: System Higraph Model: Behavior Allocation II

All of the figures in this section have shown the capabilities of higraphs to link requirements (structural and behavioral), components (attributes and functions), and behaviors (states and transitions). However, as noted all throughout this section, the real power of a higraph system model is that all of these connections exist whether they are shown in a diagram or not.

Thus, the real power of the higraph system model comes from the mathematical and logical equation, the higraph quadruple **H = (B, E, ρ, Π)**, that defines it.

Section 7.6: Mathematical and Logical Model

To construct the mathematical and logical model of the office network system, we will follow the guidelines presented in Section 2.2 and Section 5.1. After the requirements, structure, and behavior models exist and are connected, a good way to begin construction of the math model is to define the possible meanings behind each node, edge, hierarchy, and orthogonal region.

From the higraphs presented thus far in this example, our list of nodes would look as follows:

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}

Table 5: Office Network Higraph Model Node Definitions

Each row in the table above defines a set by putting a logical label on it. Each node in any part of the Office Network higraph will fall into one of these sets. For example, the set $B_{1-2-1-2}$ (System Behavior Requirements) would consist of four nodes: Power On, Power Off, Run Applications, Run Security. When the hierarchy portion of the model is defined, any nodes that fall under these four would also make up the set $B_{1-2-1-2}$.

The table below, Table 6, shows a list of all of the logical definitions applied to edges:

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	E_{5-4}

	Function	
	Satisfaction of a Domain Requirement by a System Requirement	E_6

Table 6: Office Network Higraph Model Edge Definitions

Again, each row in the table above defines the logical sets of edges that make up the Office Network higraph model. Every edge in the model falls into one of these sets. For instance, the set E_6 (Satisfaction of a Domain Requirement by a System Requirement) would consist of the three edges shown in Figure 75 that connect domain requirements to network requirements.

The reference to a "Multiplicity Association" in the table above would define the edges shown in Figure 64 in this chapter that shows how instances of system structure components associate with each other.

In a similar manner to our definitions of nodes and edges, hierarchy and orthogonality within a higraph need to be logically defined. Four this example, they are defined in the tables below:

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

Table 7: Office Network Higraph Model Hierarchy Definitions

As an example, 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.

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

Table 8: Office Network Higraph Model Orthogonality Definitions

An example of a hierarchy set would be Π_1 (Requirements Domain) that would consist of four nodes: Structural Requirements, Cost Requirements, Power Requirements, and Behavioral Requirements.

From these four logical definition tables, all nodes, edges, hierarchies, and orthogonal regions can be placed into one—**or more**—sets. A set by itself is not terribly helpful. Even if all nodes are defined and grouped according to Table 5 above, we still need to know where they "fall" in the higraph. This information comes from Tables 7 and 8 (hierarchy and orthogonality). Of course, to know how anything relates to anything else in the system, we need to know the set of edges.

Section 7.7: Using the Office Network Higraph Model

As discussed all throughout this paper, 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, based on the tables in Section 7.6 above.

For instance, if our requirement to interface the office network with a T3 WAN link 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_{1-1} (The system shall interface with a T3 WAN link). To do this, we would query the Edges set for any occurrence of B_{1-1} (The system shall interface with a T3 WAN link). From our higraph equation, and from Figure 75, we would see that there exists and edge, E_{5-1} [B_{1-1} (The system shall interface with a T3 WAN link), B_{2-1-1} (WAN Speed)]. The query would then trace up through the hierarchy to find what component the B_{2-1-1} (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_{5-1}, has a meaning of "Assignment of a Structure Requirement to a Component Attribute" as defined in Table 6. So, moving up through the hierarchy from B_{2-1-1} (WAN Speed) the query would find that B_{2-1-1} (WAN Speed) belongs to the set ρ_3 (Router) = [B_{2-1-1} (WAN Speed), B_{2-1-1} (# of

WAN Ports)]. We now know that we have to modify the Router component 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. 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.

There of course are almost infinite 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 (i.e. Finance wants to know what the total cost of the system is) or equipment obsolescence (i.e. 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 servers not only to present information, but to show and validate how the system is put together.

Chapter 8: Conclusions and Future Work

Section 8.1: Conclusions

What we have intended to document in this paper are the strong possibilities the higraph formalism holds as a systems modeling language. Higraphs allow for complete and thorough representation of system information, and allow for complete interconnection among all system components, and any level of the system hierarchy.

Higraphs can defined mathematically and logically, which clears any ambiguities from the system model, as well as allows for the system model to be "smart." A "smart" system model based on higraphs could respond to queries for specific information. The data that is 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.

Higraphs themselves are a very general formalism, but can be tailored and specified to model a wide variety of systems engineering problems.

In these ways, higraphs improve on existing system modeling languages, and are themselves a viable tool for complete system modeling.

Section 8.2 Future Work

Based on the existing Higraph work, and the ideas presented in this paper, there are a few areas of future work that would prove beneficial in moving toward a more formal Higraph based modeling language.

First, determining how to use higraphs to model time would be valuable for complete systems modeling. More work to examine how best to depict information on a traditional sequence diagram, where lifelines are used to show time progression, is one area that is covered by UML and SysML, but is only briefly examined here. In addition to a lifeline feature, we may be able to come up with a new equation to add to the higraph quadruple (making it a quintuple?) that shows timing. This equation might outline what events must occur in what sequence.

Second, though we have covered the basic structure and behavior models here, additional types of models are available (i.e. Collaboration Diagrams, the previously mentioned Sequence Diagrams) in current modeling languages. Higraphs should easily be extended to allow for these new models to be added, but this is not documented or addressed here.

Third, as mentioned in Chapter 7, higraphs could be used to model systems still under design. However, we have not explored exactly how to model the early artifacts of a system design problem such as a CONOP or architecture (from which we derive our requirements, structure, and behavior models).

Fourth, and probably most critically, the most important future work would lie in developing a software implementation of a higraph model. Any software tool that implements Higraphs would, at a minimum, have to allow the following tasks:

- Create a System Requirements Higraph from user inputs
- Create a System Structure Higraph from user inputs
- Create a System Behavior Higraph from user inputs
- Allow the user to define types of nodes, edges, hierarchies, and orthogonalities
- Allow the user to connect nodes via edges

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, like those documented in Chapter 4, could be applied to import existing artifacts (Class Diagrams, Statecharts, etc.) into a new higraph model.

More important than the interfaces any software implementation of a higraph modeling tool is the underlying data structure used in the implementation. A structure must be selected that allows for complex and lengthy traces and queries. Examples may include linked lists (with list objects as nodes and links as the edges) and databases (an entity based database might work best). Whatever the data structure, it must correctly and completely represent the equation that defines the system higraph.

Another related software tool might that would be beneficial to have developed would be a tool that could translate between UML/SysML and Higraphs. With this, existing (separate) UML/SysML diagrams could be “imported” into a higraphs. From there, someone could create the required edges to unify the system model as a single higraph.

Lastly, a software tool that translates DAGs into higraphs, and vice versa, could have some practical uses.

Bibliography

1. Austin, Mark. “ENSE 622 Requirements, Design, Trade-Off Analysis, Lecture Notes.” February 2004.
2. Austin, Mark. “Lecture Notes for ENSE 621/ENPM 641: Visual Modeling of Engineering Systems with UML.” September, 2002.
3. Bajaj, Manas. Burkhart, Roger. Etc. “Experiences Using SysML Parametrics to Represent Constrained Object-based Analysis Templates.” 7th NASA-ESA Workshop on Product Data Exchange. The Workshop for Open Product & System Lifecycle Management. April 2005.
4. Bell, Alex. “Death by UML Fever.” ACM Queue, Volume 2, Number 1 (2004): <http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=130&page=1>
5. Berkenkotter, Kirsten. “Using UML 2.0 in Real-Time Development; A Critical Review.” 2003. < http://www-verimag.imag.fr/EVENTS/2003/SVERTS/PAPERS-WEB/04-Berkenkoetter-UMLRT-critic.pdf>
6. Black, Paul. "Directed Acyclic Graph", from Dictionary of Algorithms and Data Structures, Paul E. Black, ed., NIST. <http://www.nist.gov/dads/HTML/directAcycGraph.html>
7. Dupagne A. and Teller A. “Hypergraph Formalism for Urban Form Specification”, COST C4 Final Conference, Kiruna, September 21-22, 1998.

8. Grossman, Ornit. Harel, David. “On the Algorithmics of Higraphs.” Technical Report CS97-15, The Weizmann Institute of Science, Rehovot, Israel, 1997
9. Harel, David. “On Visual Formalisms.” Communications of the ACM 31 (1988): 514-530.
10. Harel David. “Statecharts: A Visual Formalism for Complex Systems.” . Science of. Computer. Programming, 1987. Vol. 8, pp. 231-274.
11. Headway Software Inc. “Closed Loop Development with Headway ReView.” June 2001. <http://headwaysoftware.com/pdf/whitepaper.pdf>
12. Headway Software, Inc. <http://headwaysoftware.com/about/customers.php>
13. Letelier, Patricio. “A Framework for Requirements Traceability in UML Projects.” In Proc. of 1st International Workshop on Traceability in Emerging Forms of Software Engineering. In conjunction with the 17th IEEE International Conference on Automated Software Engineering (2002): 32-41
14. Microsoft Corportation. Microsoft Office Visio Stadard, 2003. CD ROM. 2003.
15. Minas M. and Viehstaedt G., DiaGen: “A Generator for Diagram Editors providing Direct Manipulation and Execution of Diagrams.” IEEE Symposium on Visual Languages, Darmstadt, Germany, pp. 203-210.
16. Minas M. “Hypergraphs and a Uniform Diagram Representation Model.” In Proc. 6th International Workshop on Theory and Application of Graph Transformations (TAGT, 98), Paderborn, Germany, 1998.
17. Munzner, Tamara. “Interactive Visualization of Large Graphs and Networks.” Diss. Stanford University, 2000.

18. The Object Management Group, Inc. “Unified Modeling Language Specification, Version 1.4.2.” July 2004. <http://www.omg.org/docs/formal/04-07-04.pdf>
19. The Object Management Group, Inc. “Unified Modeling Language Superstructure, Version 2.0.” August 2005. <http://www.omg.org/docs/formal/05-07-04.pdf>
20. The Object Management Group, Inc. “What is OMG-UML and Why Is It Important?” 1997. <http://www.omg.org/news/pr97/umlprimer.html>
21. Paige R.F. “Heterogeneous Specifications and their Application to Software Development." Research Proposal, Department of Computer Science, University of Toronto", August 1995.
22. Ramaswamy M., and Sarkar S. “Using Directed Hypergraphs to Verify Rule-Based Expert Systems.” IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 2, March-April 1997, pp. 221-237.
23. Rational Software Corporation, Microsoft Software Corporation, etc. “UML Summary, Version 1.1.” September 1997. <http://umlcenter.visual-paradigm.com/umlresources/summ_11.pdf>
24. SysML Partners. “Systems Modeling Language Specification, Version 0.9 DRAFT.” January 2005. < http://www.sysml.org/artifacts/specs/SysML-v0.9-PDF-050110.zip>
25. SysML Partners. “Systems Modeling Language Specification, Version 1.0 alpha.” November 2005. < http://www.sysml.org/artifacts/specs/SysMLv1.0a-051114R1.pdf>

26. Telelogic, AB. Telelogic DOORS. <http://www.telelogic.com/corp/products/doors/doors/index.cfm>.

27. UML Forum. "UML FAQ." <http://www.uml-forum.com/faq.htm#What%20changes%20are%20expected%20in%20UML%202.0,%20and%20when%20is%20it%20planned>

28. UML Glossary. Computer Science Department, California State University, San Bernardino. June 2005. <http://www.csci.csusb.edu/dick/samples/uml.glossary.html#F%20G%20H>

29. Wikipedia. "Graph Theory." <http://en.wikipedia.org/wiki/Graph_theory>

30. Wissen, M. and Ziegler, J. "A Methodology for the Component-Based Development of Web Applications." In Stephanidis, C.; Jacko, J. (Eds.): Proceedings of 10th Int. Conf. on Human-Computer Interaction (HCI International 2003), Vol. 1, Crete, Greece, 2003