ABSTRACT

Title of Document:	SYSTEM MODELING EXAMPLES USING HIGRAPH FORMALISMS
	Jason E. Smith Master of Science, Systems Engineering, 2007 University of Maryland, College Park
Directed By:	Dr. Mark Austin Department of Civil and Environmental Engineering and Institute for Systems Research University of Maryland, 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 system models are detailed and comprehensive, but lack a true ability to unify the system model by showing all relationships among all components in the model. This paper shows by example how 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. This allows for a “smart” model that can be queried for custom sets of information that will aid a systems engineer in engineering decisions.

SYSTEM MODELING EXAMPLES USING HIGRAPH FORMALISMS

By

Jason E. Smith

Scholarly paper 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
2007

Advisor:
Dr. Mark Austin

Table of Contents

Introduction

System engineers are constantly searching for the perfect modeling tool that addresses every need: easy to use and understand, detailed, flexible, logical, consistent, and verifiable. Although much research has gone into finding the perfect tool even the most widely recognized modeling languages are routinely criticized. The current leading language for visual modeling, Unified Modeling Language (UML) and it's domain specific systems engineering subset Systems Modeling Language (SysML) are often attacked for being too large and complex, utilize imprecise semantics, without specification, are inflexible to future change, and lack the ability to display relationships among components and diagrams [6]. Kevin Fogarty's master's thesis titled *System Modeling and Traceability Application of the Higraph Formalism* attempts to show how one criticism; traceability can be eliminated with the use of a mathematical graph know as higraphs [1]. As seen in his thesis and briefly in this paper, higraphs can help provide a true unified system model with a rational and visible relationship between components, requirements, and models themselves. They all but eliminate traceability concerns. Although higraphs are not an independent perfect solution, nor a self supporting modeling tool, they can dramatically improve system modeling when incorporated with existing languages. This paper is intended to address one area of future work in Fogarty's thesis: to create a complete system model using higraphs that includes the basic structure and behavior models, as well as other lesser used models such as sequence diagrams [1]. To complete this objective this paper provides some introductory information on the weaknesses of current modeling languages, the basic theory behind higraphs, and a detailed modeling and analysis example of a University Registration System.

Disadvantages of existing languages

A visual modeling language is an artificial language comprised of a predefined set of diagrams, notations, and principles used to increase the ease of understanding a system, similar to a blueprint for a construction engineer. These languages are applied in various disciplines, including business process modeling, computer science, information management, software engineering, and systems engineering. Regardless of the application, they all have the ability to be used to specify system requirements, structures and behaviors. The goal of the language is to visually or textually represent a system so that throughout its lifecycle, all stakeholders (i.e. customers, operators, analysts, and designers) can better understand the system, its subsystem, and relationships among subsystem entities. Although existing visual modeling languages such as UML and SysML are detailed and comprehensive, they lack a true ability to display all relationships among components within the model. A specific weakness is that they rarely allow for flows of information between the diagrams. Without a visible relationship, it is impossible to create an automated trace from a requirement, to a component, to a behavior, to a test case. This results in separate disjointed models and never a complete unified system.

Higraph Basics

Higraphs are a type of mathematical graph that combines depth and orthogonality. The concept of higraphs as a graphical representation was first introduced by David Harel in his 1988 ACM paper titled, *On Visual Formalisms* [4]. In this paper he describes higraph as a formalism based on Euler circles and Venn diagrams with connections. Harel starts with a traditional Venn diagram as seen in Figure 1.

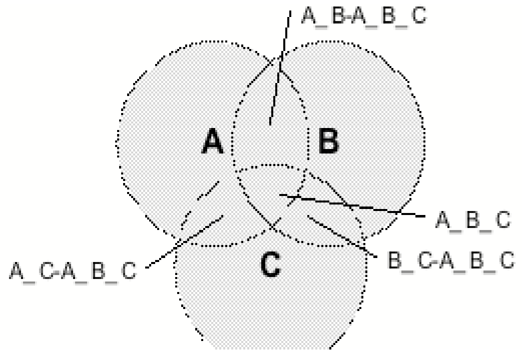


Figure 1: Simplistic Venn diagram

In the Venn diagram the notation of enclosure, inclusion, and exclusion defines the relationship between each set. Harel deviates from the standard Venn diagram by creating a unique contour for every set, intersection of sets, and exclusion of sets found in the Venn diagram [4]. When necessary, these unique contours are grouped together to show enclosure or a hierarchy. These unique contours, commonly referred to as “blobs”, are created to easily identify without confusion or lengthy equations what is and what is not included in each entity [4]. For example the entity labeled B∩C-A∩B∩C in the Venn diagram found in Figure 1 is represented simply as blob L in the higraph representation found in Figure 2.

The most low level blobs in a graph are known as an “atomic blob”. Atomic blobs represent real identifiable sets containing no wholly enclosed blobs within them. Any blob, other than an atomic blob, denotes the compound set consisting of the union of all blobs that are totally enclosed within it [4]. The atomic blobs of Figure 2 are G, H, I, J, K, L, and M. Although not all are at the same level they all represent one distinct set. The remaining blobs in Figure 2, A, B, C, D, E, and F are a compound set of other blobs and therefore not considered an atomic blob.

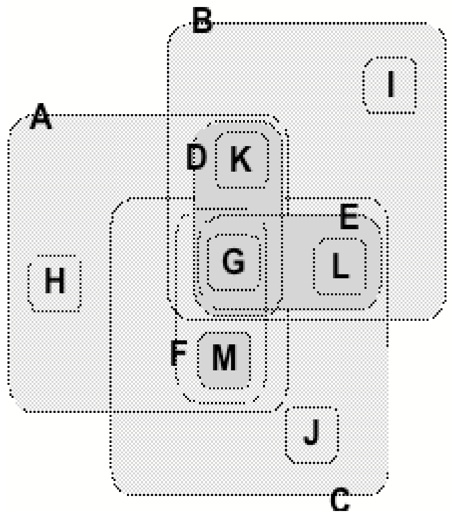


Figure 2: Basic higraph

The lack of an atomic blob within an intersection does not in itself have meaning. In Figure 3, blob M (intersection of blob A and C less the intersection of blob A, B, and C) from diagram 2 has been removed. This indicates that blob F (intersection of blob A and blob C) now has the same meaning as blob G even though blob F is physically extended further. Unless internal blobs are present within an intersection, the junction is meaningless. The only assumption one might draw is that blob F is a space saver for a blob that might be filled at some time in the future.

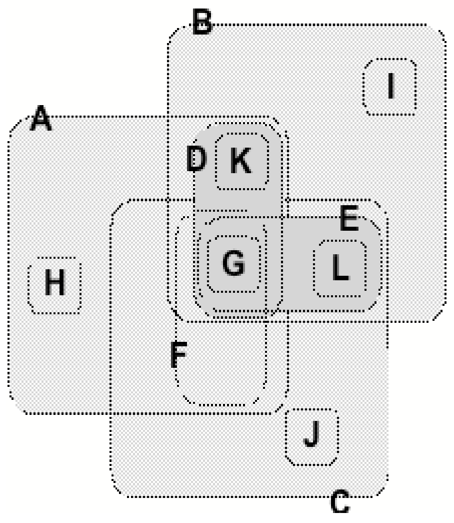


Figure 3: Basic highraph with empty blobs

A second way to create independence between entities is by partitioning blobs with dashed lines to crate an unordered and associative “orthogonal component” [4]. This representation is no different than the contour of a blob in theory, however it offers a different visual display that may prove beneficial in some cases. An example utilizing orthogonal lines is presented in Figure 4.

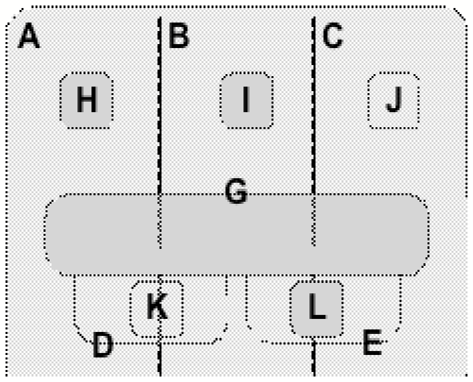


Figure 4: Basic higraph with orthogonal lines (not representing blobs F and M)

Finally, edges are attached to the contour of any blob to define various relationships (physical, logical, etc.) between blobs that can not be easily visualized [4]. Edges can be directed or undirected, labeled or unlabeled, and connect one blob to itself (self-directed), another, or many other blobs. An example using edges is provided in Figure 5.

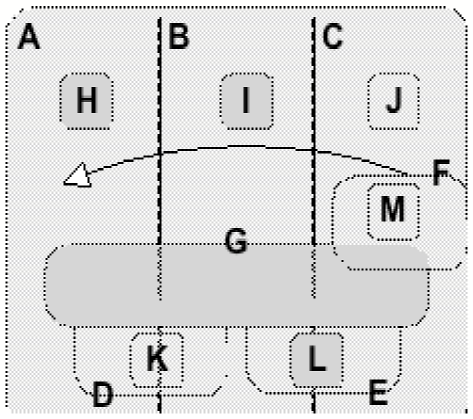


Figure 5: Basic higraph with orthogonal lines and edges

Much of the transformation from a traditional Venn diagram to higraph discussed so far is concentrated on "smart grouping". An additional benefit to higraphs is their mathematical properties which are thoroughly defined in Harel's follow on paper titled *On the Algorithmics of Higraphs*. The basic mathematical definition of a higraph can be summarized as follows [3]:

- 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)
- H is the highraph quadruple (B, E, ρ, Π)

In Harel's follow on paper, he describes in detail the syntax and semantics of higraphs. By creating notations and mathematical tenets for higraph blobs, edges, and orthogonal lines, all relationships such as generalization, hierarchy and inheritance can be mathematically shown, even for the most complex systems. These mathematical properties increase a highraph's ability to logically connect elements even further than its visual representation. One of the greatest benefits of these mathematical properties is the ability for custom sets of information to be logically ordered and queried.

These mathematical characteristics also enable higraphs to be easily used in supporting software tools. The characteristics provide the ability to create, display, and query general connectivity and organizational relationships. Modeling software that utilizes higraphs is already available. Headway Software has employed higraph modeling with their source code visualization tool, reView. It's promoted as the first visual tool to intelligently show all dependencies at all levels of development including application, package, class, method, and data members [5]. The use of this tool in software development shows promise for the systems engineering field. Software tools like reView are proving that higraph modeling is a great platform for visualizing and understanding the overall system structure and hierarchy.

<u>**Higraphs as a system modeling language**</u>

For all the above mentioned reasons (visual representation, mathematical properties, and software adaptability), higraphs can be useful in systems modeling, particularly for traceability. In many respects the concept of higraphs is the basis for statecharts and is currently used for modeling systems. In fact

UML’s state diagram is the principal diagram used to define system behaviors. Fogarty takes the concept one step further by adapting higraph’s in every aspect of system modeling, thereby improving UML and SysML diagrams. They can duplicate the graphical advantages of existing diagrams while providing traceability. Additionally, they can be used to represent dynamic and static systems as shown in Figure 6.

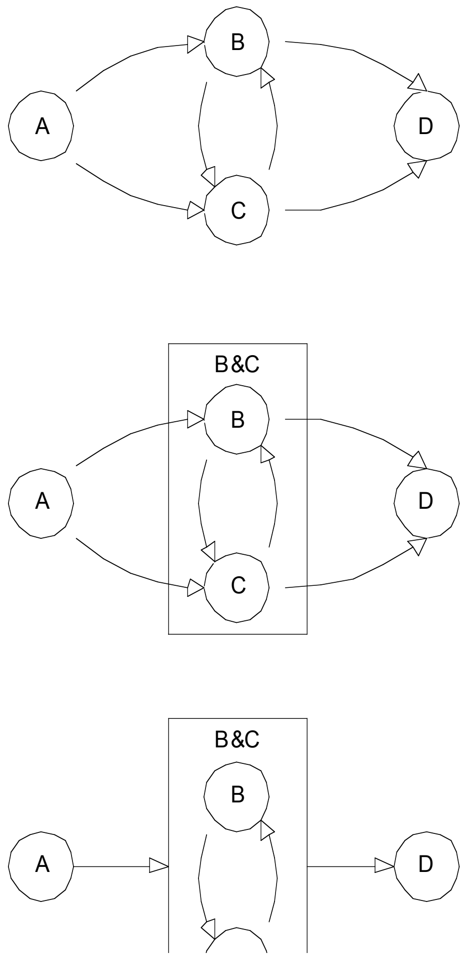


Figure 6: Higraph representation of a finite state machine

In brief and for the purposes of this paper, every object, even use cases within a system have three elements; requirements, structure, and behavior. The details that make up these elements often have relationships or connections within the object/use case as well as within elements outside the object/use case. Examples of relationships include associations, dependencies, extensions, generalizations, includes, realizations, and transitions. Figure 7 shows the most high level higraph representation of an object. In the figure the object itself is displayed as a blob with sub blobs representing the object's elements. Elements and details within the elements can overlap or be connected by an edge to show a relationship, typically a commonality. In reality, even for simple systems there could be thousands upon thousands of blobs all interconnected making for an unusable graph. It is important to balance the amount of information that can be put on the graph with the amount of information the user needs on the graph. Generalizing diagrams with respect to the user's view point by minimizing the amount of sub-blobs is critical to creating an effective graph. The user should realizes that further levels of detail exist which are either of no importance to their viewpoint or can be queried if needed.

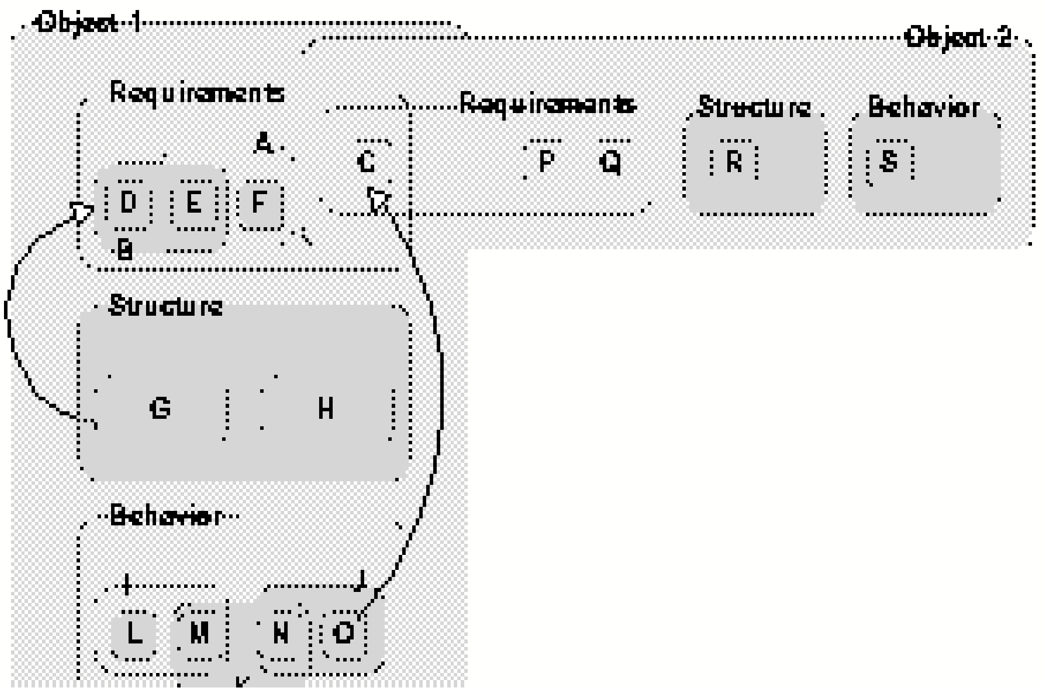


Figure 7: Basic system engineering modeling using higraph

The remainder of this paper concentrates on an example used to identify and show the potential advantages higraphs bring over traditional models. The example used is a University Registration System and addresses the system's three elements: requirements, structure, and behavior.

University Registration System Example

The purpose of this chapter is to demonstrate how higraphs can be applied to the representation and organization of system modeling elements. The ability for higraphs to be used as a modeling tool is demonstrated through the detailed modeling of an existing University Registration System. The first section of this chapter concentrates on incorporating higraph concepts into use case development and use case diagrams. The next three sections present the system separately, modeling each entity independently, requirements, structure, and behavior. The final section demonstrates how the higraph allows for flows of information between the diagrams and entities.

System Organization

The example focuses on three high level university administrative use cases including university application, course registration, and transcript request. Actors in the system include students (future students, current students, and alumni), instructors, coordinators, bursar, and registrar. As seen in Figure 8, academic actors such as students, instructors, and coordinators fall under a college/school and department while managerial actors such as bursar and registrar fall under the administration. Other objects such as transcripts, courses, and sections are represented in order to update, verify, or record information. Detailed discussion on class diagrams and their higraph equivalent are provided in the structure section of this chapter. Since the goal of this example is to highlight the interconnectivity relationships and thoroughness higraphs have over traditional modeling languages, some of the less relevant visual modeling development processes, such as requirement generation and use case description have been omitted.

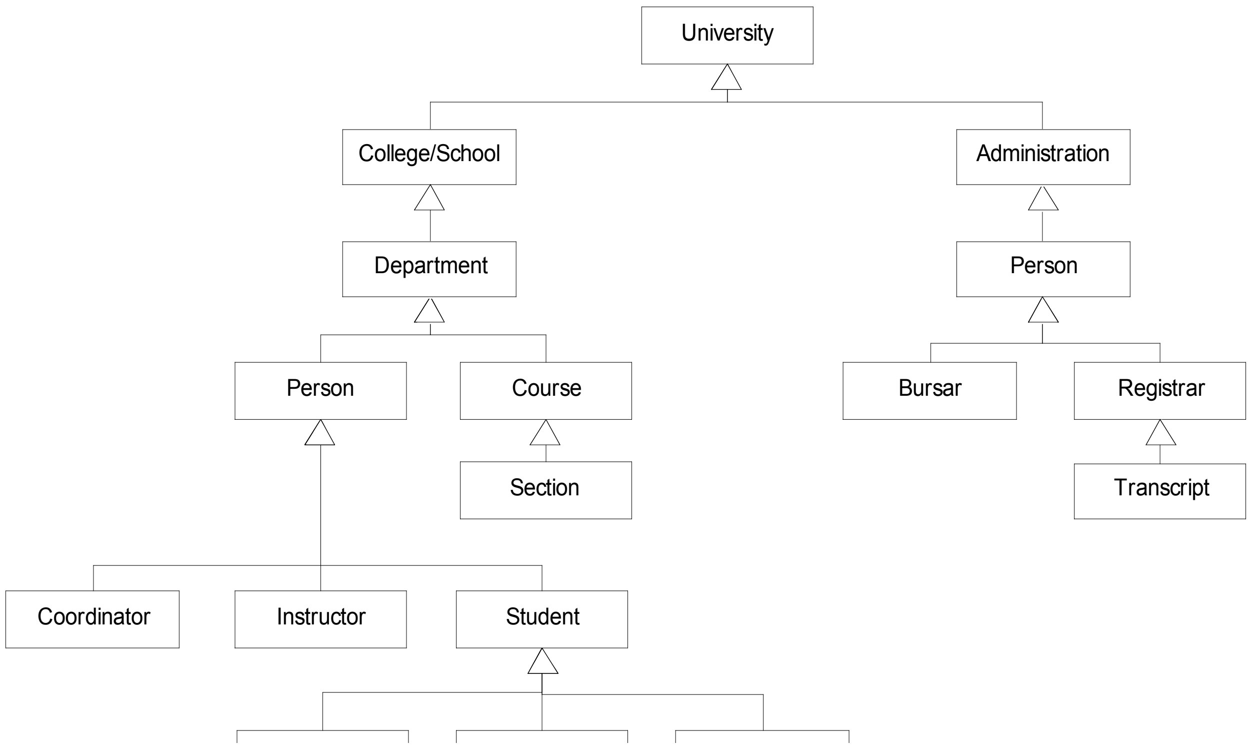


Figure 8: Preliminary concept hierarchy for the University Registration System

Use Cases and Scenarios

One of the first steps in modeling a system is to generate use case diagrams, use cases, and textual scenarios. The use case diagram is a simple graphical model for representing the primary relationships within a system. It is used to identify the primary entities (people and things that achieve results) and processes that form the system. The primary entities that interact with the system are termed "actors" and the processes or functions are called "use cases." The use case diagram shows which actors interact with what use case. Complex engineering systems are often described with families of use case diagrams. A specific use case diagram might represent system functionality from a specific viewpoint or interest. Of course, these viewpoints will be linked. Any provision needs to be made for extension of functionality and proper treatment of irregular functionality (when something goes wrong). To handle these relations, UML specifies three types of relationships between use cases. The three relationship (include, extend, and generalization) are defined as follows;

- the included use case is one step of the initial use case
- the extended use case is the next step after the initial use case
- generalized use case is a specialized form or sub-class of the initial use case

In an actual model each use case and scenario would be defined with text in finer detail. Since this process is not improved or simplified by the use of hiraphs, it is omitted from this paper.

As shown in the examples below, higraphs can be used to represent use cases. Higraphs ability to provide an unambiguous graphical representation and logical traceability significantly improves the traditional UML use case diagram. The following figures show the transformation from use case diagram to higraph at three distinct levels within the system. The first set of use case diagrams and higrahps represent the complete system at its highest level where the second and third set of use case diagrams and higraphs represent one use case from the previous level (mid level: course registration and low level: enroll student). At each level a UML use cases diagram is juxtaposed with a highraph equivalent containing the same information. Figure 9 graphically shows how the use case diagrams and higrahps are presented. Once all three levels are discussed an inclusive use case higraph is presented to truly demonstrate how higraphs can significantly improve use case diagram's visual and traceability characteristics.

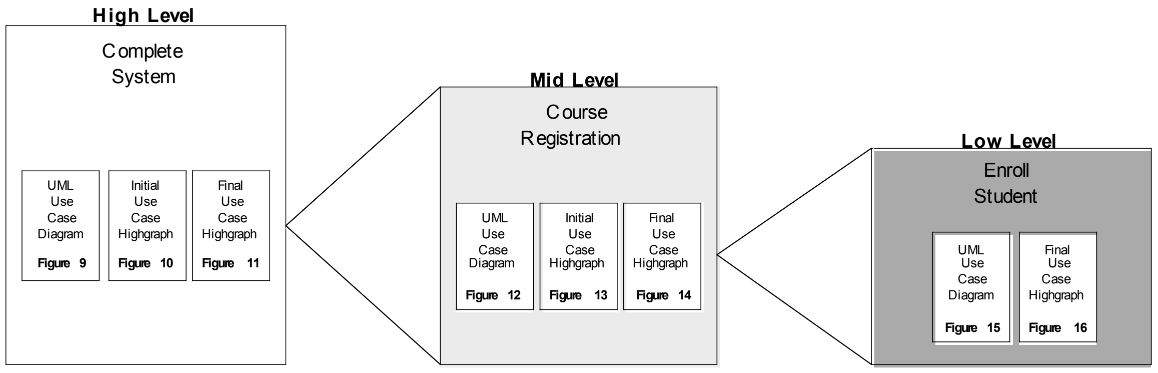

High Level
Complete System
UML Use Case Diagram Figure 9
Initial Use Case Highgraph Figure 10
Final Use Case Highgraph Figure 11
Mid Level
Course Registration
UML Use Case Diagram Figure 12
Initial Use Case Highgraph Figure 13
Final Use Case Highgraph Figure 14
Low Level
Enroll Student
UML Use Case Diagram Figure 15
Final Use Case Highgraph Figure 16

Figure 9: Graphical representation of how use case diagrams and use case higraphs are presented in this section.

High Level Use Case Example

The use case examples are first presented from a high level complete system perspective. At this level, Figure 10 shows the traditional UML approach to visually modeling use cases.

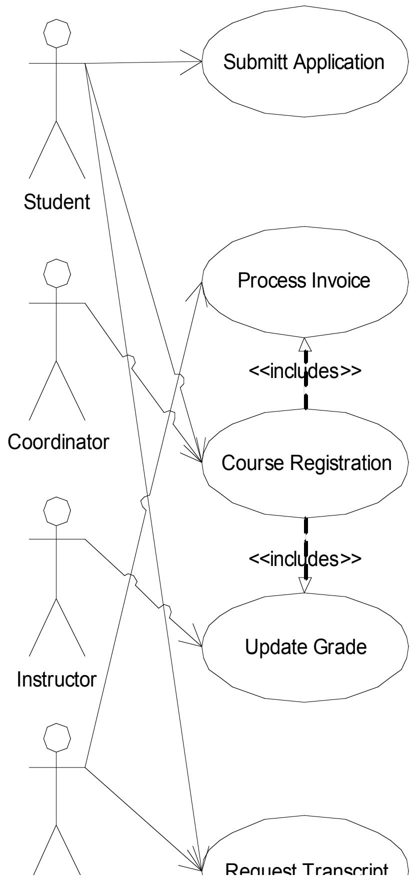


Figure 10: UML high level use case diagram representing the complete system

The two higraph equivalents shown in Figure 11 and 12, display the evolution in higraph form. The first higraph (Figure 11) looks and feels very much like a use case diagram with higraph aspects and the second higraph (Figure 12) looks and feels like a true higraph.

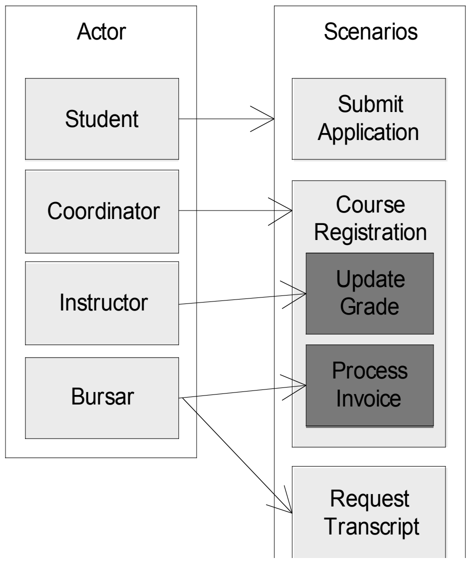


Figure 11: Initial high level use case higraph representing the complete system

One flaw with the initial use case higraph is that use case relationships such as included, extended, or generalized are not properly represented. For example, in both the UML use case diagram (Figure 10) and initial use case higraph (Figure 11), the student is shown to interact with all scenarios. In fact the student does interact with all scenarios but not all parts of all scenarios. Using higraphs to represent use cases can provide the ability to employ sub blobs that clearly illustrate the relationships between and within objects and scenarios. As shown in the high level final use case higraph (Figure 12), sub blobs can be successfully used to avoid this common confusion and create a logical and traceable graphical representation.

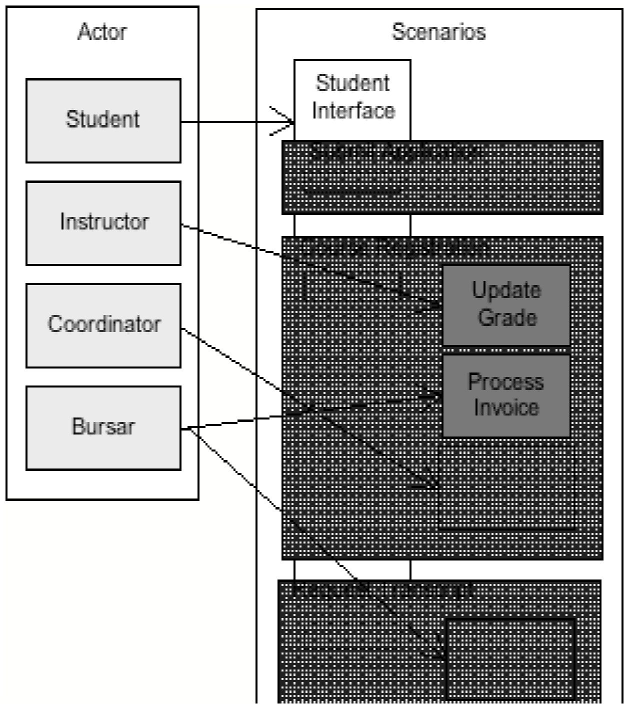


Figure 12: Advanced high level use case higraph representing the complete system

Mid Level Use Case Example

The next set of use case examples are presented from a mid level perspective representing the course registration scenario. Again the example begins with the traditional UML use case diagram shown in Figure 13.

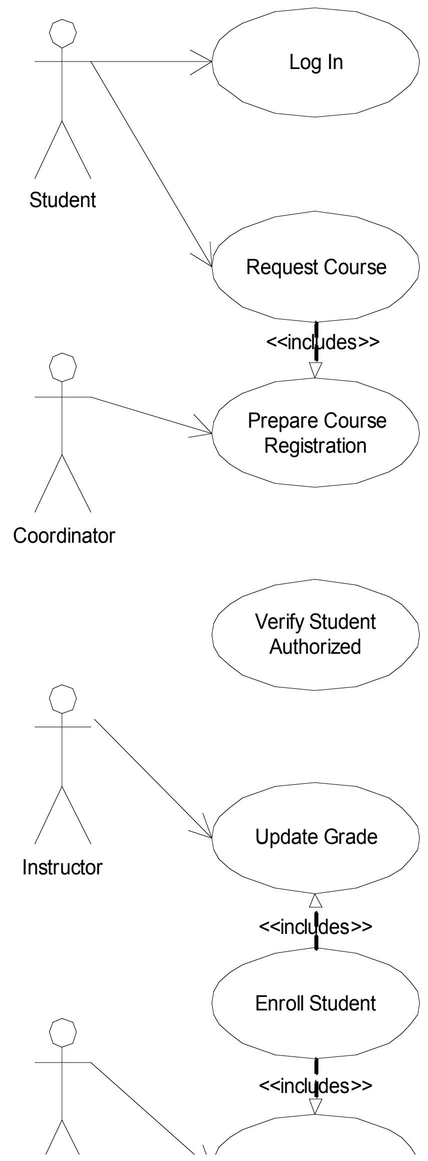


Figure 13: UML mid level use case diagram representing the "*course registration*" use case

As illustrated in the high level example, the higraph equivalents are shown in two figures to display the evolution from use case diagram to use case higraph. The first higraphs in Figure 14 are extremely similar to their use case diagram counterpart. The reason for two higraphs in Figure 14 is to show the benefits of properly utilizing blob hierarchy in all aspects of the higraph. The left hand higraph is clearly more difficult to follow when compared to the right hand higraph. The right hand higraph was easily simplified by generating a blob titled "course preparation team" which eliminated two unnecessary edges. This type of effective hierarchy management significantly improves the overall quality and usability of any higraph.

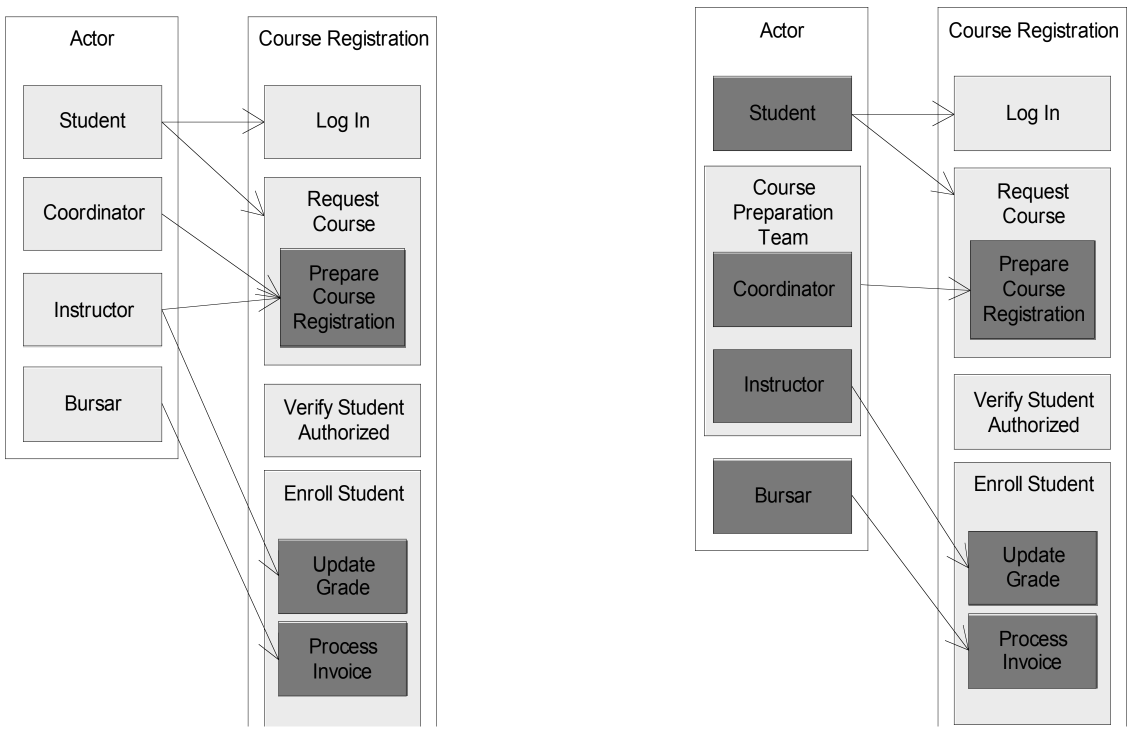


Figure 14: Initial mid level use case higraph representing the *"course registration"* use case

Once again the initial higraph representation does not properly represent the use case relationships. For example, in Figure 13 and 14, the student is shown to interact with the "request course" use case but it is difficult to determine if the student interacts with the included "*prepare course registration*" use case. In reality, it is the coordinator and instructor who prepare the course registration, not the student. To avoid this confusion, the final higraph in Figure 15 utilizes sub blobs to more accurately represent exactly what part of the use case the user interacts with.

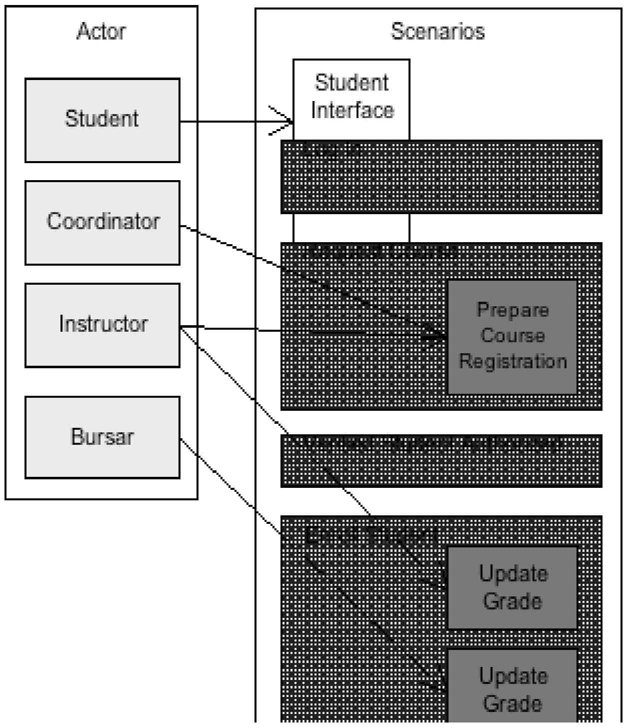


Figure 15: Advanced mid level use case higraph representing the *“course registration”* use case

Low Level Use Case Example

The final set of use case examples are presented from a low level perspective representing the enroll student scenario from the course registration use case. As with the two previous use case examples, the enroll student example begins with a traditional UML use case diagram shown in Figure 16.

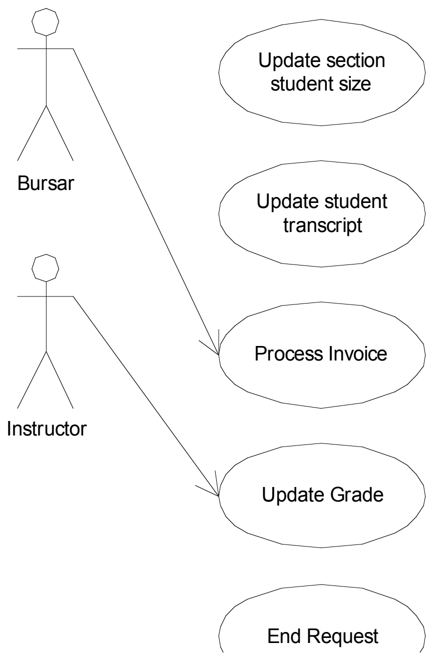


Figure 16: UML low level use case diagram representing the "*course registration*" "*enroll student*" use case

Unlike the high and mid level examples, the enroll student example is especially straightforward. Due to this simplicity, there is little change between the use case diagram in Figure 16 and the final use case higraph in Figure 17. For this reason an initial use case higraph and discussion regarding the evolution between the two higraphs is not necessary.

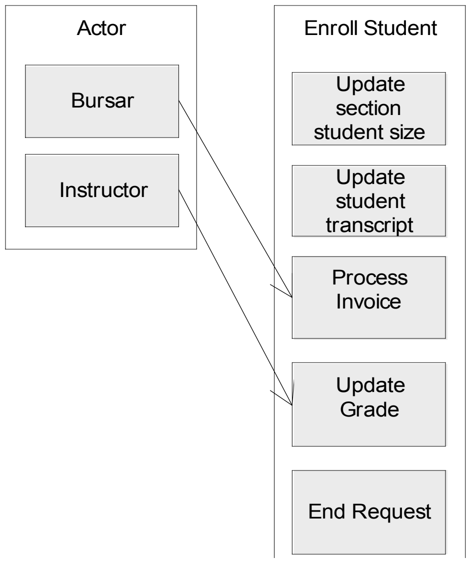


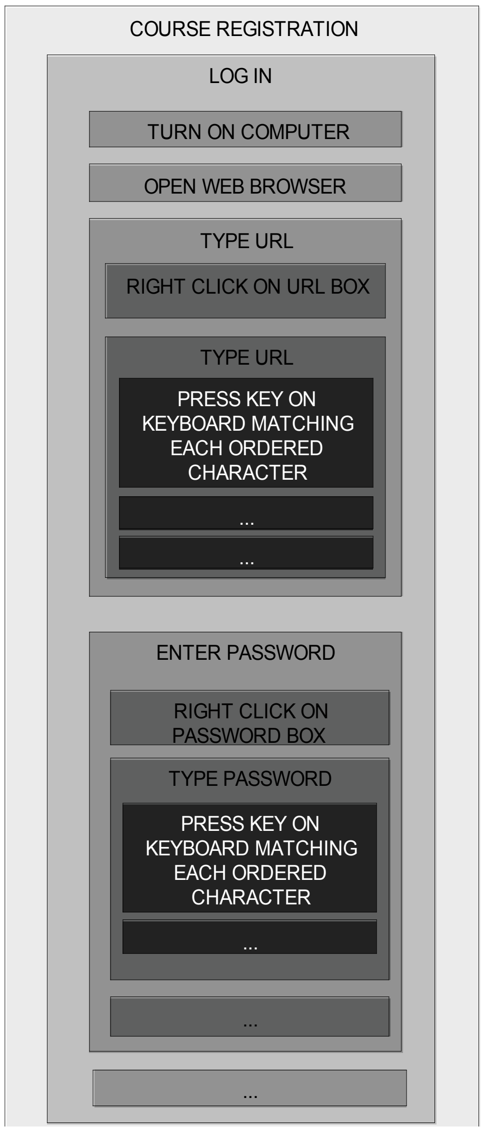
Figure 17: Final low level use case higraph representing the “course registration” “enroll student” use case

The advantages of use case higraphs are apparent: by organizing both actors and use cases into a hierarchy of higraphs, the number of relationships can be compressed and simplified, without compromising meaning in the diagram. This presentation is considerably more logical, effective, and useful.

Inclusive Use Case Higraph

The last several examples have shown how higraphs can effectively and clearly visualize all relationships within a use case. The next step in the development of use case higraphs is to demonstrate that all relationships throughout different use case higraphs can be represented in one master higraph. This master higraph can be thought of as an inclusive use case higraph.

Based on higraph theory and its application for systems engineering models, a blob describes particulars of an object or use case at a particular level of detail. As discussed above, a blob is considered atomic if it does not contain any sub blobs within it. Like most blobs, use case blobs can always be subdivided into an almost infinite number of lower levels, eventually reaching a scenario or literal atomic level. For instance, the left hand higraph in Figure 18 the use case course registration includes many steps, each at differing levels of detail; beginning with *turning on the computer* to *how to type in individual characters on the keyboard*. Use case blobs can overlap or use edges to indicate commonalities. It is important to note that use case blobs are not necessarily listed sequentially, although in particular situations order may make the graph easier to read. When order is necessary, use of edges is recommended over overlap to indicate commonalities. The right hand higraph in Figure 18 displays the use of edges to simplify use case blobs.



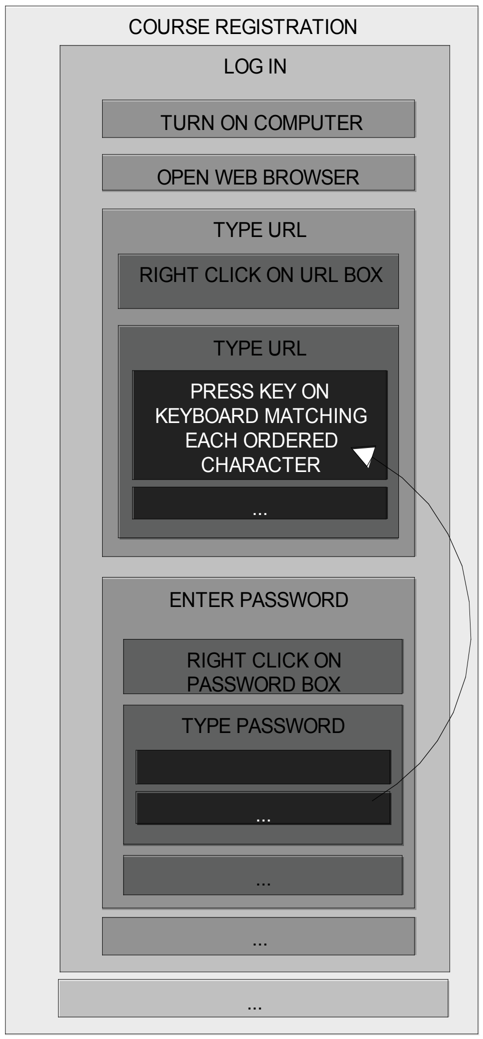


Figure 18: Complex and simplified representation of use case higraph

Utilizing this concept, one can create an inclusive use case higraph to represent all levels of interactions between every actor and use case within a system. Figure 19 represents an inclusive use case higraph for the University Registration System at the predefined high, mid, and low levels.

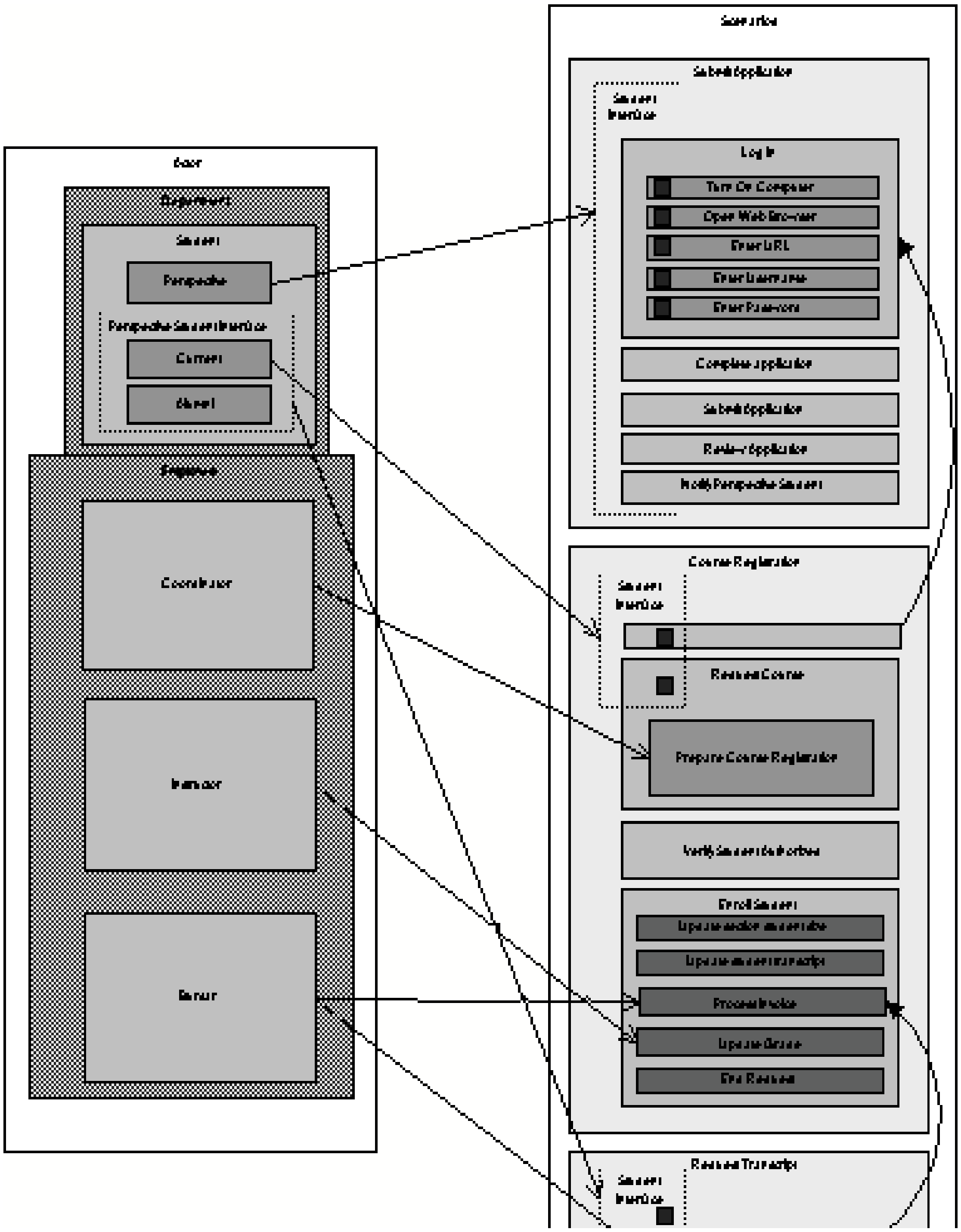


Figure 19: Inclusive use case higraph representing the complete University Registration System

As discussed earlier in this paper, it is important to balance the amount of information that can be put on the graph with the amount of information the user needs on the graph. Providing too much information can quickly turn an effective graph, particularly a large graph such as an inclusive use case higraph, ineffective. To avoid this pitfall, inclusive use case higraphs can and should be shaped to fit the user's viewpoint by eliminating unnecessary information. Viewpoints can be made from one or many actor perspectives, one or many use case perspectives, or some combination of the two. Figure 20 shows how a simplified inclusive use case higraph might look from the student perspective.

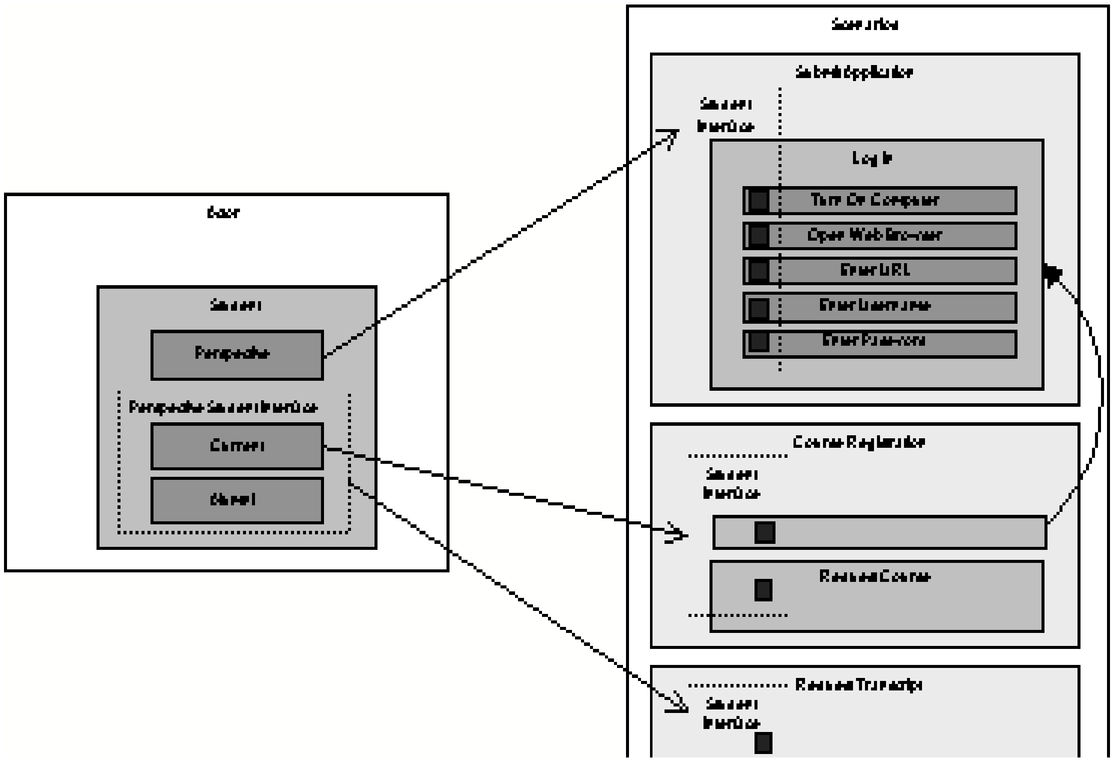


Figure 20: Inclusive use case higraph from student perspective

This type of use case representation has utility beyond the traditional use case diagram and in some respects resembles activity and sequence diagrams without decision nodes. This characteristic provides initial proof that higraphs possess the ability to display relationships among components and diagrams. In any case, the concept may be useful but its details require further study.

System Requirements

With the system use cases in place, designers then need to define details of the functionality specified in the desired use cases. Typically this functionality is referred to as the system's functional requirements. These functional requirements specify the internal workings and particular behaviors of the system such as the calculations, technical details, data manipulation, and processing. Functional requirements are supported by non-functional requirements, which impose limitations on the design or implementation. Non-functional requirements, also know as constraints, specify the overall characteristics or performance requirements such as cost, reliability, security, quality standards, and design constraints. In general terms they can be thought of as adjectives used to describe the behaviors of functions specified by the functional requirements. All requirements have similar relationships as use cases and objects. For example, requirements can be "a part of" or "the same as" another requirement or "inherited by" or "dependant on" an object or use case. Figure 21 shows a generalized description of how these relationships can be represented in higraph form. In the figure a non-functional requirement is associated with a functional requirement which is associated with a use case.

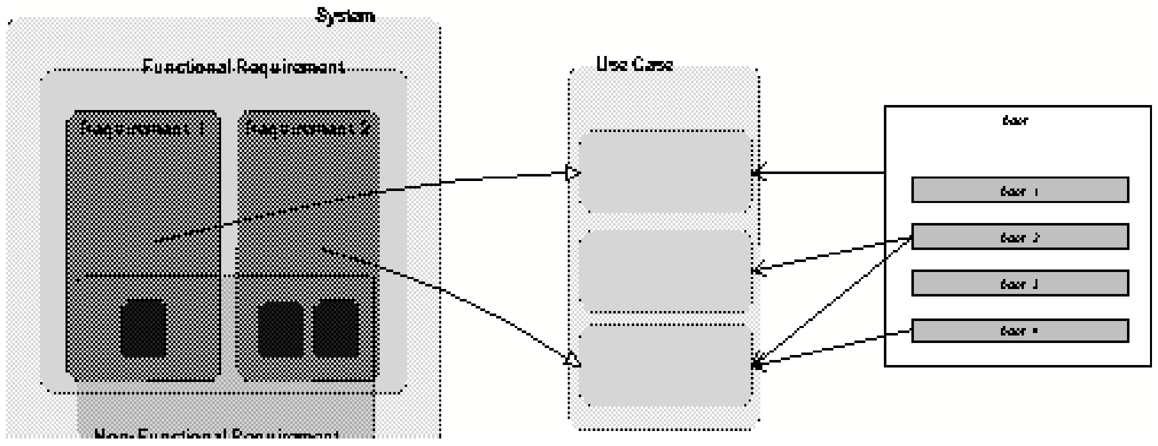
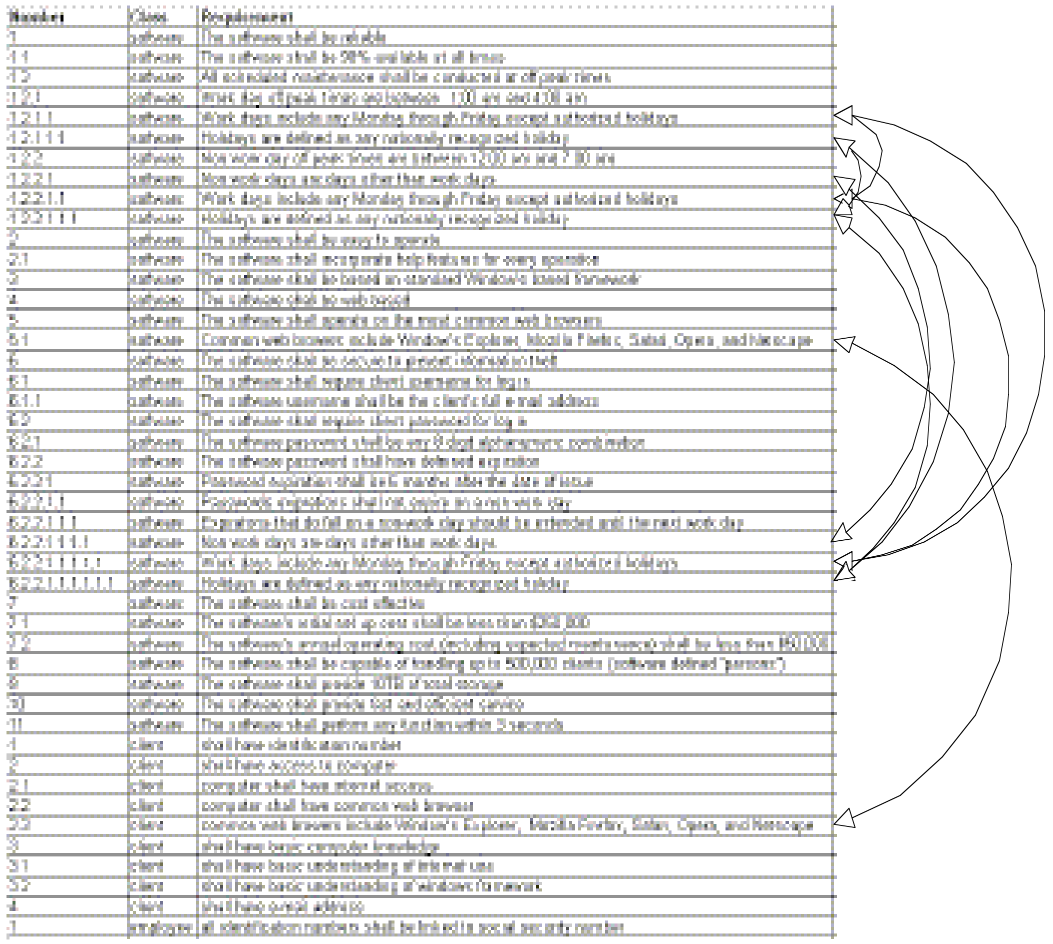


Figure 21: Generalization of requirement relationships in higraph form.

In addition to the text of a requirement such as “must be in English and Spanish”, requirements must contain a unique name, number, and rationale. This information is used to help the reader understand why the requirement is needed, and to track the requirement through the development of the system. Without these details, requirements can not be validated or verified nor could current and future designers be able to determine why a particular requirement exist or if it is even necessary.

Based on this brief requirement introduction, tracking requirements is the key to an effective system. UML does not provide for requirements diagrams, and often creates rather antiquated requirements lists or databases with little or no visual representation or relationship links as shown in Figure 22.



Number	Class	Requirement
1	software	The software shall be reliable
1.1	software	The software shall be 99% available at all times
1.2	software	All scheduled maintenance shall be conducted at off peak times
1.2.1	software	Work day off peak times are between 1:00 am and 4:00 am
1.2.1.1	software	Work days include any Monday through Friday except authorized holidays
1.2.1.1.1	software	Holidays are defined as any nationally recognized holiday
1.2.2	software	Non work day off peak times are between 12:00 am and 7:00 am
1.2.2.1	software	Non work days are days other than work days
1.2.2.1.1	software	Work days include any Monday through Friday except authorized holidays
1.2.2.1.1.1	software	Holidays are defined as any nationally recognized holiday
2	software	The software shall be easy to operate
2.1	software	The software shall incorporate help features for every operation
3	software	The software shall be based on standard Window's based framework
4	software	The software shall be web based
5	software	The software shall operate on the most common web browsers
5.1	software	Common web browsers include Window's Explorer, Mozilla Firefox, Safari, Opera, and Netscape
6	software	The software shall be secure to prevent information theft
6.1	software	The software shall require client username for log in
6.1.1	software	The software username shall be the client's full e-mail address
6.2	software	The software shall require client password for log in
6.2.1	software	The software password shall be any 8 digit alphanumeric combination
6.2.2	software	The software password shall have defined expiration
6.2.2.1	software	Password expiration shall be 6 months after the date of issue
6.2.2.1.1	software	Password expirations shall not expire on a non work day
6.2.2.1.1.1	software	Expirations that do fall on a non-work day should be extended until the next work day
6.2.2.1.1.1.1	software	Non work days are days other than work days
6.2.2.1.1.1.1.1	software	Work days include any Monday through Friday except authorized holidays
6.2.2.1.1.1.1.1.1	software	Holidays are defined as any nationally recognized holiday
7	software	The software shall be cost effective
7.1	software	The software's initial set up cost shall be less than $[illegible]
7.2	software	The software's annual operating cost (including expected maintenance) shall be less than $[illegible]
8	software	The software shall be capable of handling up to 500,000 clients (software defined "persons")
9	software	The software shall provide 10TB of total storage
10	software	The software shall provide fast and efficient service
11	software	The software shall perform any function within 3 seconds
1	client	shall have identification number
2	client	shall have access to computer
2.1	client	computer shall have internet access
2.2	client	computer shall have common web browser
2.3	client	common web browsers include Window's Explorer, Mozilla Firefox, Safari, Opera, and Netscape
3	client	shall have basic computer knowledge
3.1	client	shall have basic understanding of internet use
3.2	client	shall have basic understanding of windows framework
4	client	shall have e-mail address
1	employee	all identification numbers shall be linked to social security number

Figure 22: Simplistic requirements list showing relationship links

Smart Grouping

A coherent higraph requirement representation can be easily visualized using the same “smart grouping” techniques employed to create the use case graphs. Requirements graph’s improvements over the requirements list are apparent. The graph logically shows relationships between requirements and objects constrained by the requirement. Derived and generalized requirements as well as details regarding who owns or is impacted by the requirement are represented by hierarchies and edges. The following two Figures 23 and 24, demonstrate how requirements from the University Registration System example can be more efficiently displayed as higraphs. The first example, Figure 23 illustrates how the requirement “common web browsers shall include Explorer, Firefox, Safari, Opera, and Netscape” is common or “same as” for both system and user requirements. This relationship is quickly made apparent by the use of an edge.

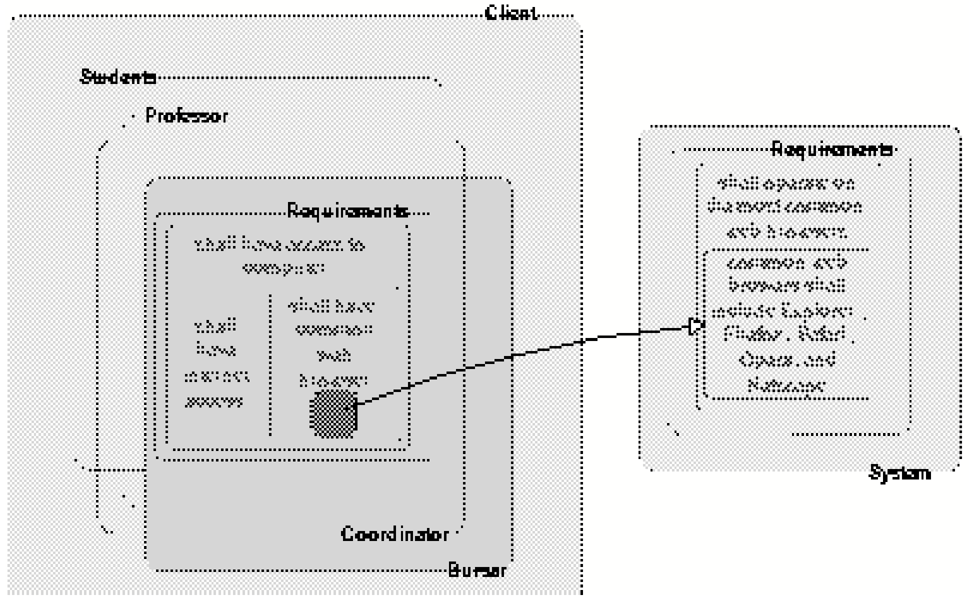


Figure 23: Higraph Requirement Diagram showing commonalities between client requirements and system requirements

The second example, Figure 23 illustrates how the requirements “non work days are days other than work days”, “work days include any Monday through Friday except authorized holidays”, and “holidays are defined as any nationally recognized holidays” are related. These requirements take on either “same as” or “inherited by” relationships between other requirements within the system requirements. The relationships would not be apparent without the use of edges. This type of visual representation in Figures 23 and 24 clearly indicates to the reader that changes made to any requirement with edges will effect other requirements which are linked.

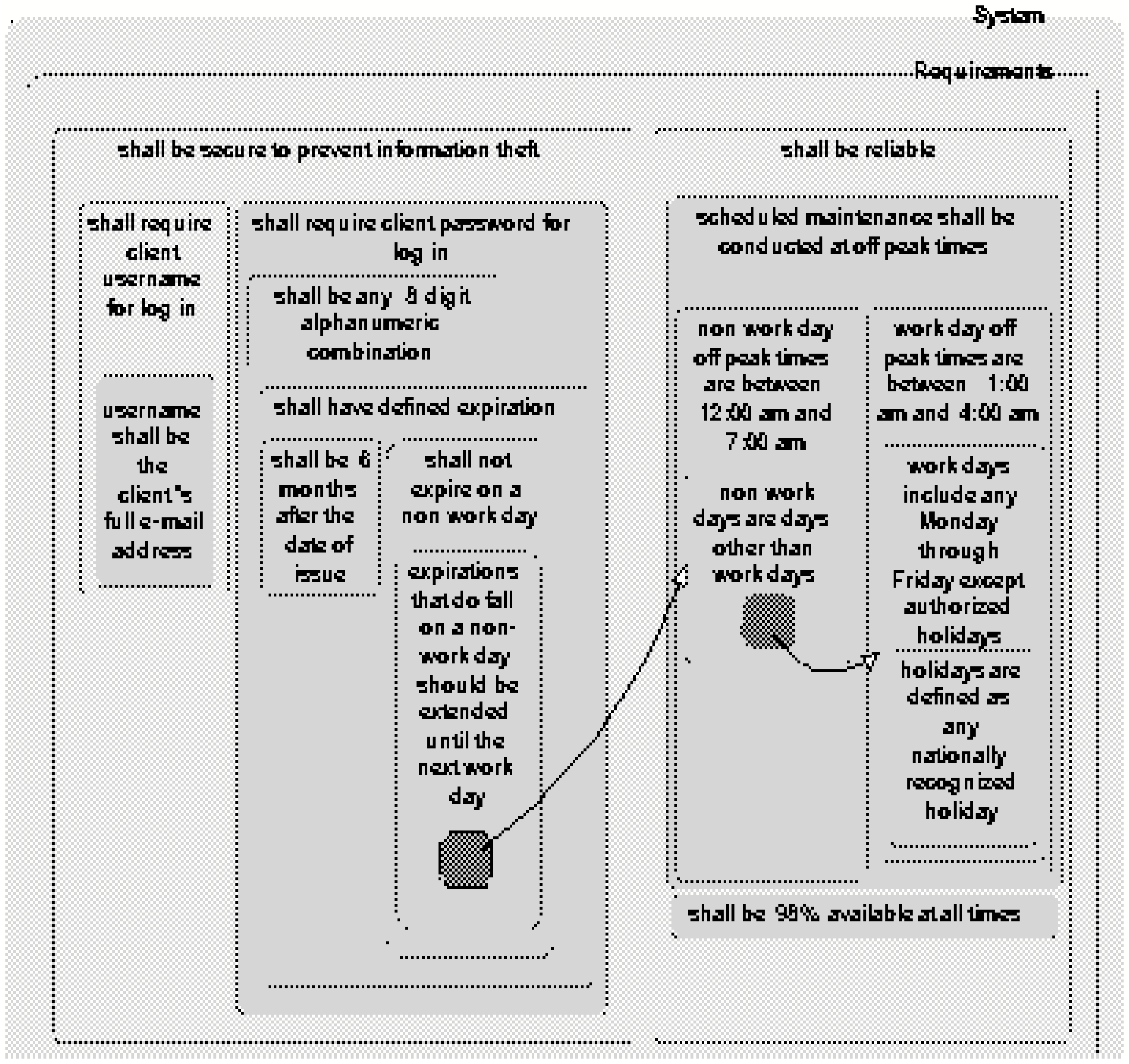

System
Requirements
shall be secure to prevent information theft
shall require client username for log in
username shall be the client's full e-mail address
shall require client password for log in
shall be any 8 digit alphanumeric combination
shall have defined expiration
shall be 6 months after the date of issue
shall not expire on a non work day
expirations that do fall on a non-work day should be extended until the next work day
shall be reliable
scheduled maintenance shall be conducted at off peak times
non work day off peak times are between 12:00 am and 7:00 am
non work days are days other than work days
work day off peak times are between 1:00 am and 4:00 am
work days include any Monday through Friday except authorized holidays
holidays are defined as any nationally recognized holiday
shall be 98% available at all times

Figure 24: Higraph Requirement Diagram showing commonalities between student requirements and employee requirements

Unlike UML, SysML contains a requirements diagram, as shown in Figure 25, which effectively displays requirements hierarchy and can create trace relationships between requirements and other elements. The requirements diagram in Figure 25 illustrates how the University Registration System's requirement for client ID numbers would be represent in SysML. This type of requirements modeling provides the additional benefit of traceability and provides the purpose for each requirement.

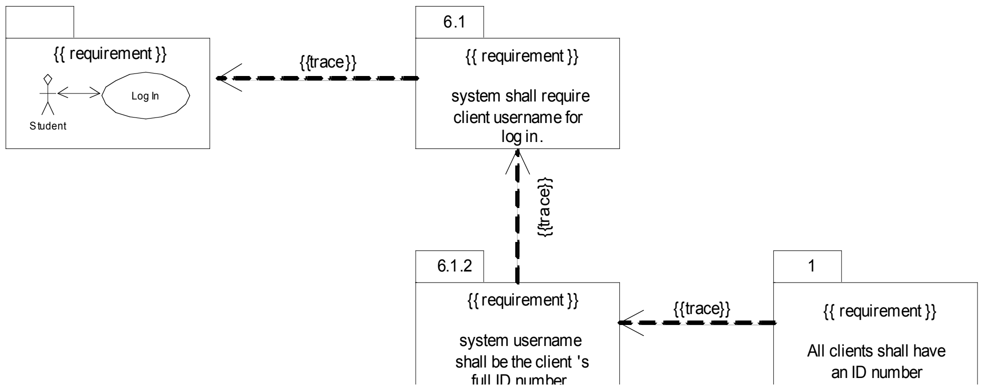

{{ requirement }}
Log In
Student
{{trace}}
6.1
{{ requirement }}
system shall require client username for log in.
{{trace}}
6.1.2
{{ requirement }}
system username shall be the client 's full ID number
{{trace}}
1
{{ requirement }}
All clients shall have an ID number

Figure 25: Sample SysML Requirements Diagram

This advantage makes SysML's requirement diagram a better model for a requirement higraph. Using the same example as Figure 25, Figure 26 demonstrates how a requirements diagram can be modeled using higraph formalisms. This inclusive requirement graph again shows smart grouping techniques within the requirements themselves. The use of edges indicates their relationship to or inheritance from use cases. The use of higraphs for requirement modeling again proves that higraphs can best represent all relationships among components within the system.

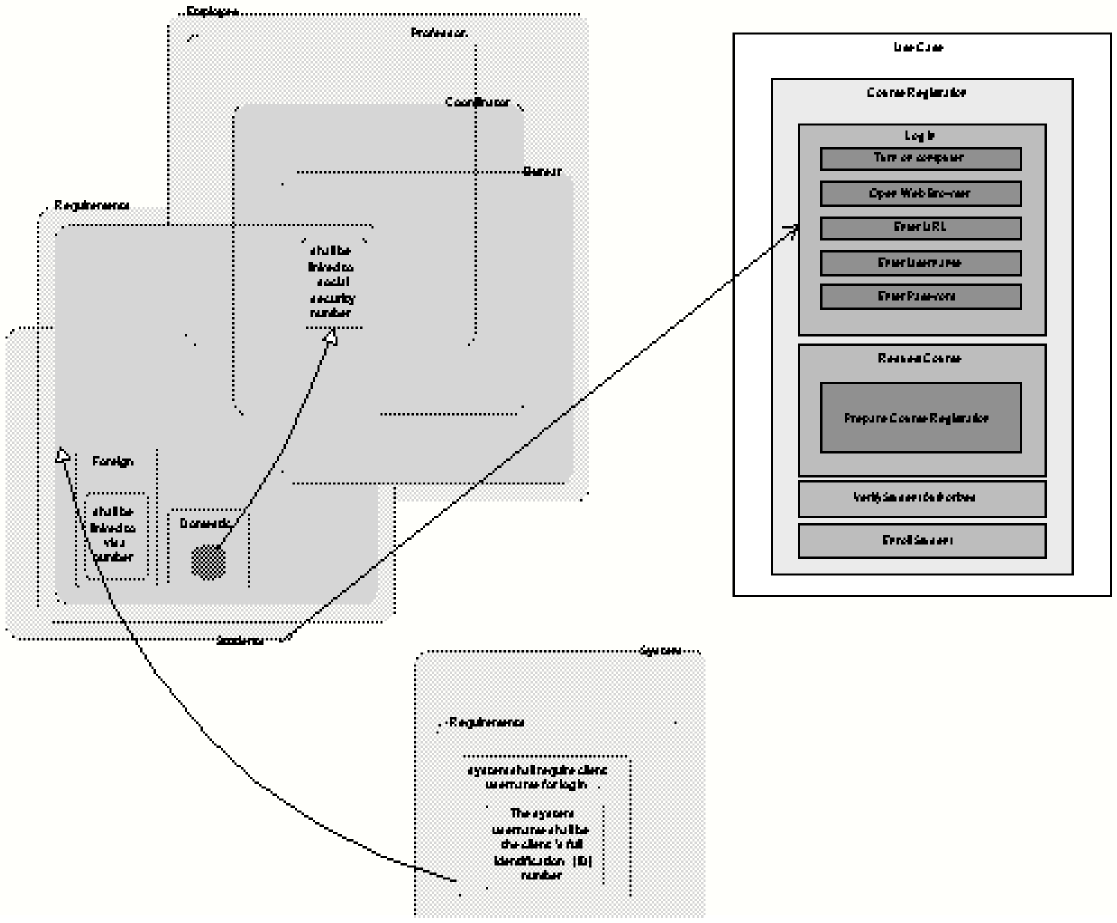

Employee
Professor
Coordinator
Bursar
Requirements
shall be linked to social security number
Foreign
shall be linked to visa number
Domestic
Students
Use Case
Course Registration
Log In
Turn on computer
Open Web Browser
Enter URL
Enter Username
Enter Password
Request Course
Prepare Course Registration
Verify Student Qualification
Enroll Student
System
Requirements
system shall require client username for log in
The system username shall be the client's full identification (ID) number

Figure 26: Requirement Higraph showing commonalities within system requirements

System Structure

The next step in system development is to specify how the system will accomplish its purpose. The system structure corresponds to collections of interconnected objects and subsystems, constrained by the environment within which the system must exist. The nature of each object/subsystem will be captured by its attributes and operations. The purpose of structure diagrams is to show the static structure of the system being modeled. UML and SysML structure diagrams include the class or static structure, component, composite structure, deployment, object and package diagrams. For the purpose of this paper we will concentrate on the class/static structure diagrams since they form the foundation of object-oriented analysis and design.

Class Diagram

From a top-down development perspective, system-level models begin to take shape by identifying the system structure with the use of UML use cases or SysML static structure diagrams. In the same style of previously discussed diagrams, the class diagram describes the structure of the system by showing the classes of the system, the relationships between classes (both instance level and class level), and the operations and attributes of the classes. In class diagrams, instance level relationships are in most cases synonymous with thephrase “has a” and includes association, aggregation, and composition. Alternatively, a class level relationship is synonymous with the phrase “is a” and includes generalization (14). Another distinction found on a class diagram is that instance level relationships may also detail the number of instances or objects that participate in the relationship. This concept is known as multiplicity and is typically categorized into one of the following types;

0...1	No instances, or one instance
1	Exactly one instance
0...*	Zero or more instances
1…*	One or more instances

A very basic class diagram for concepts in the University Registration System was presented earlier in this paper, Figure 8. We now expand it in Figure 27 to include additional class attributes, operations, and instance level relationships.

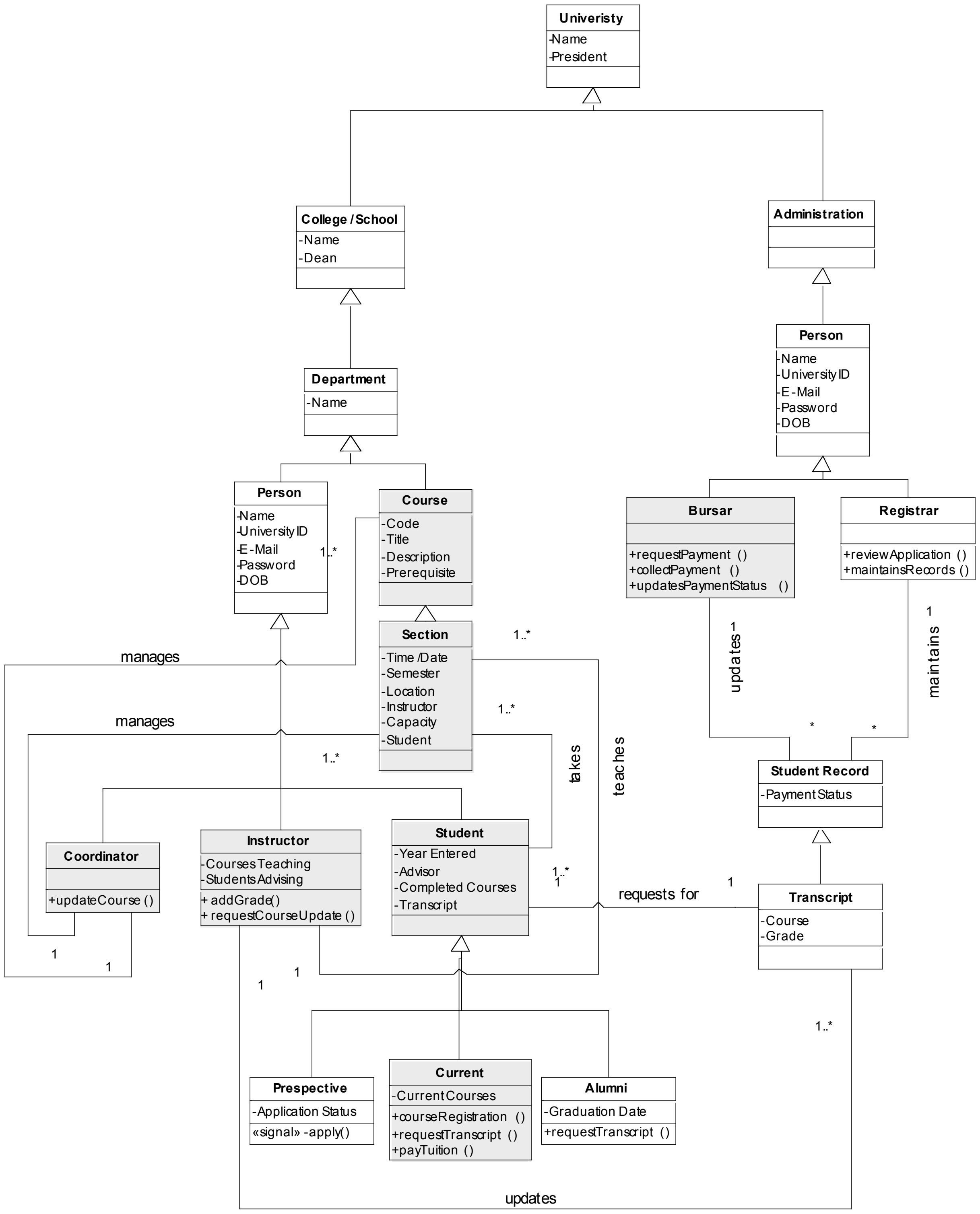

Univeristy
-Name
-President
College / School
-Name
-Dean
Administration
Department
-Name
Person
-Name
-University ID
-E-Mail
-Password
-DOB
Course
-Code
-Title
-Description
-Prerequisite
Section
-Time /Date
-Semester
-Location
-Instructor
-Capacity
-Student
Bursar
+requestPayment ()
+collectPayment ()
+updatesPaymentStatus ()
Registrar
+reviewApplication ()
+maintainsRecords ()
Student Record
-Payment Status
Transcript
-Course
-Grade
Coordinator
+updateCourse ()
Instructor
-Courses Teaching
-Students Advising
+ addGrade()
+ requestCourseUpdate ()
Student
-Year Entered
-Advisor
-Completed Courses
-Transcript
Prespective
-Application Status
«signal» -apply()
Current
-Current Courses
+courseRegistration ()
+requestTranscript ()
+payTuition ()
Alumni
-Graduation Date
+requestTranscript ()
manages
manages
takes
teaches
updates
maintains
requests for
updates
1..*
1..*
1..*
1..*
1..*
1..*
1
1
1
1
1
1
1
1
*
*

Figure 27: UML Class Diagram with attributes, operations, and instance level relationships. Classes in the University Registration System are shown in the context of the university system, college/schools, and administrative services.

In Figure 28, the same diagram was updated to focus on those classes that are directly related to the Universal Registration System. In this last update, unnecessary classes were eliminated.

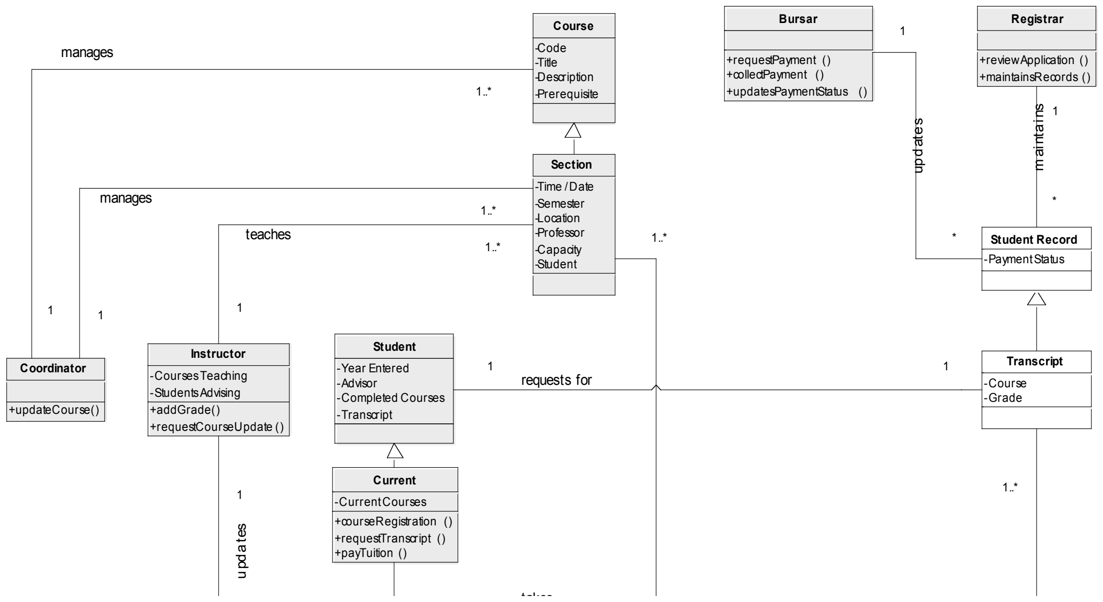


Figure 28: UML Class Diagram with unnecessary classes eliminated

The characteristics of higraphs make it easy to transform from a class diagram to a structure higraph. The same progression made with the class diagrams above are made with structure higraphs in Figures 29, 30, and 31. Starting with Figure 29, a higraph representation of the simplistic class diagram (Figure 8) shows in many respects the communicative and logical advantages of a structure higraph.

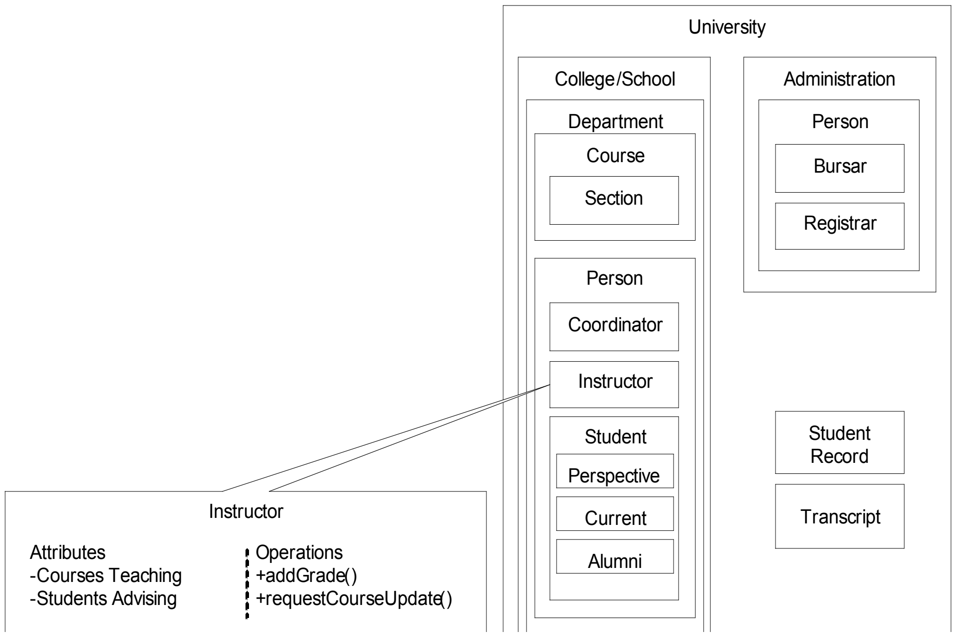


Figure 29: Simplistic Structure Higraph

The structure higraph in Figure 30 demonstrates how additional class details (attributes, operations, and instance level relationships) can be added without loosing the higraph's visual advantages. When compared to its structure diagram counter part in Figure 27, Figure 30 is visually more comprehensible.

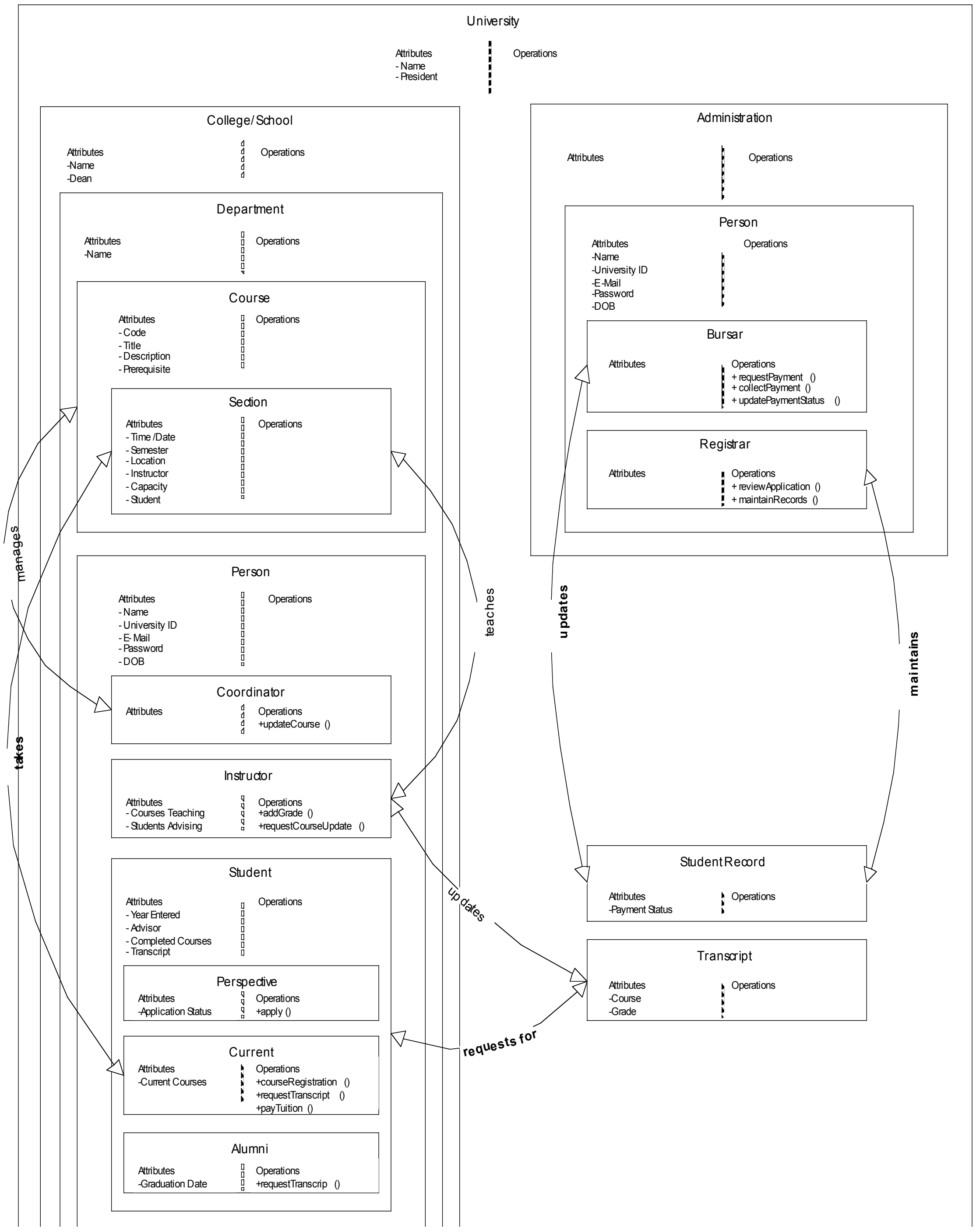


Figure 30: Structure Higraph with attributes, operations, and instance level relationships

In this last transformation the structure higraph, shown in Figure 31, is resized to illustrate only those classes link to the University Registration System. This new structure higraph is more communicative than its structure diagram equivalent in Figure 28 and now more succinct than its predecessor in Figure 30.

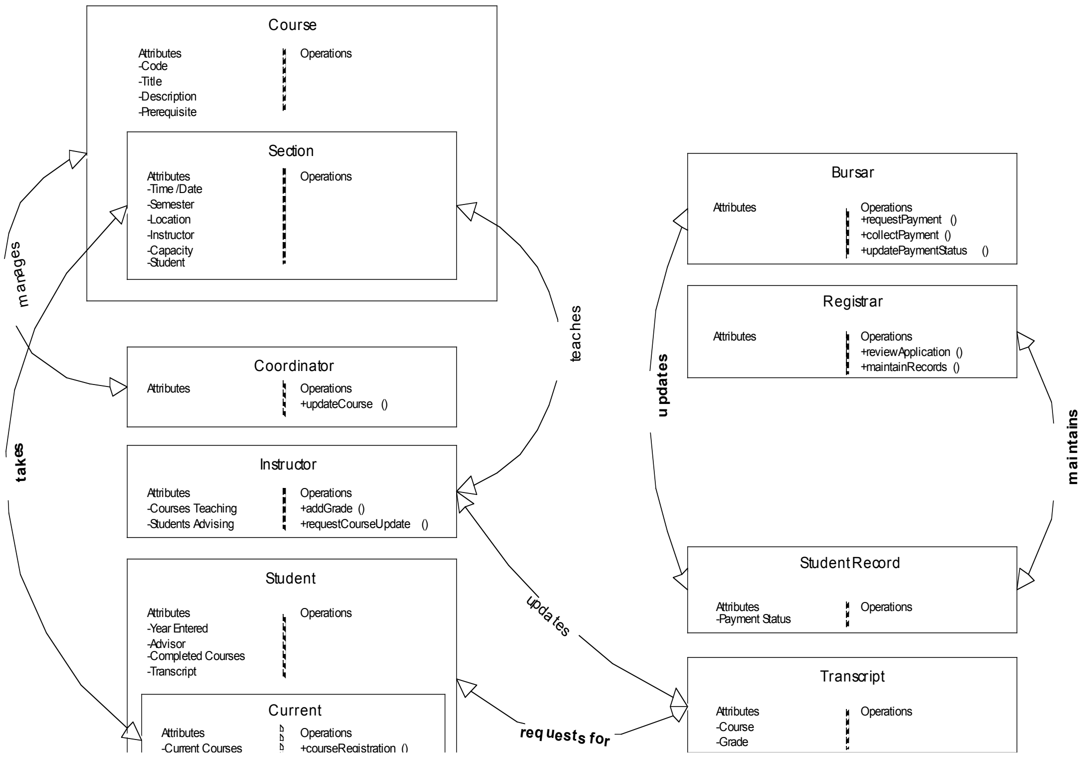


Figure 31: Structure Higraph with unnecessary classes eliminated

A structure higraph's ability to visually depict all relationships in a communicative and logical form is not found with traditional UML or SysML diagrams. A particular strength of the structure higraph is its representation of class level relationships. The graph representation clearly visualizes the delineation of classes and offers more definition. Figure 32 illustrates the class level advantages that structure higraphs offer over class diagrams. In these Figures the distinction between male/female, professors/students/coordinators/ unclassified, and university/non-university classes is apparent. It is easy to see all the class combinations that exist. For example, all professors are grouped in blob Q which consists of all male professors (blob U) and all female professors (blob V). The atomic blobs A, B, C, and D are the lowest level blobs representing professors and define whether or not the professor is a male or female, and whether that professor is part of the university system or is external to the university system. As mentioned previously, the lack of an atomic blob within an intersection does not in itself have meaning. The fact that blob R (all students) intersects with blob W (all non-university) on both sides of the graph but has no blobs contained within the intersections, indicates that there are no personnel representing this classification; students not internal to the university. Similarly, blob S (all coordinators) and blob W (all non-university) do not intersect on either side of the graph which also indicates that there are no personnel representing this classification; coordinators not internal to the university.

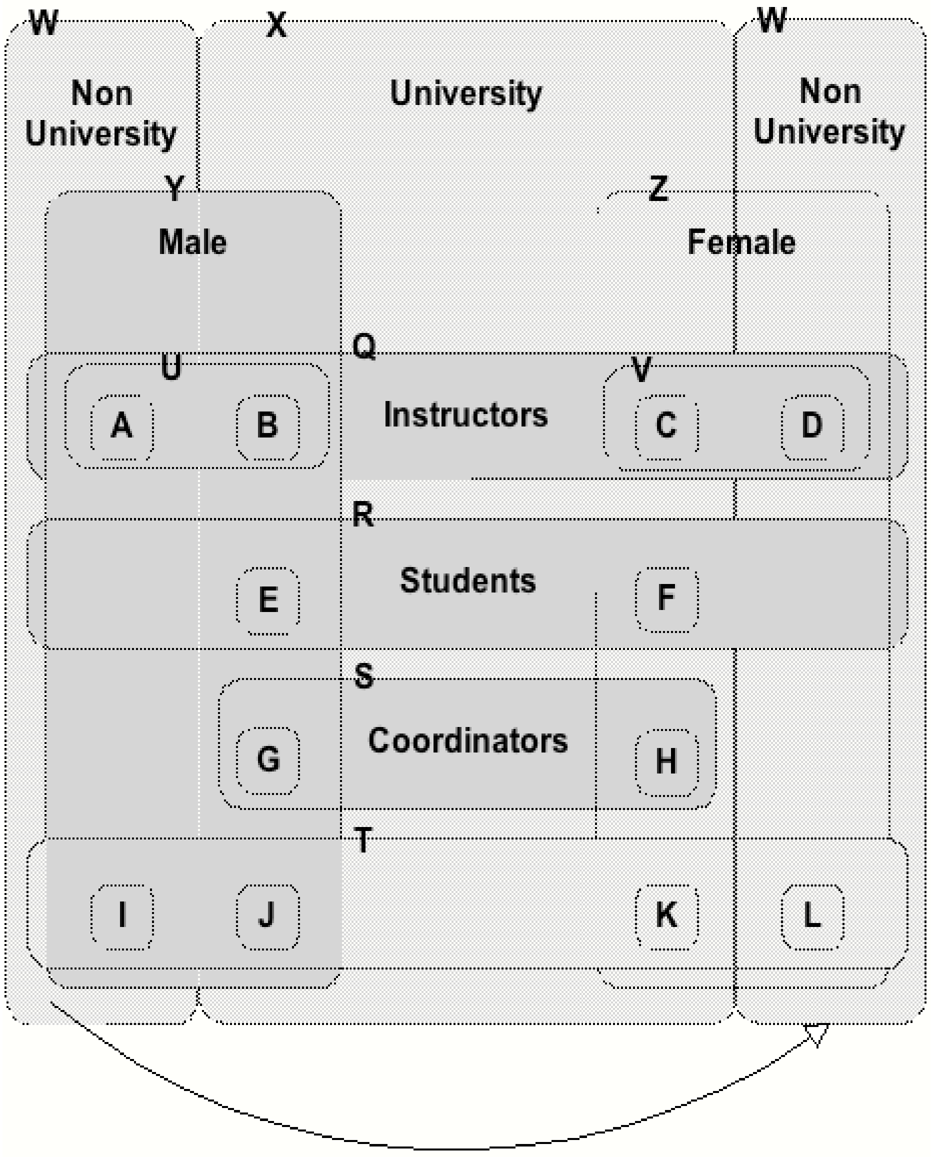


Figure 32: Higraph representing class level relationships

List and description of atomic classes

A	all male instructors not part of the university system (ie. guest lecturers, visiting professors, etc)
B	all male instructors part of the university system
C	all female instructors not part of the university system (ie. guest lecturers, visiting instructors, etc)
D	all female professors part of the university system
E	all male students part of the university system
F	all female students part of the university system
G	all male coordinators part of the university system
H	all female coordinators part of the university system
I	all male personnel not elsewhere classified which are part of the university system (ie. janitors, human resource personnel, sports director, etc.)
J	all male personnel not elsewhere classified which are not part of the university system (ie contract workers)
K	all female personnel not elsewhere classified which are part of the university system (ie. janitors, human resource personnel, sports director, etc.)
L	all female personnel not elsewhere classified which are not part of the university system (i.e. contract workers)

System Behavior

System behavior specifies what the system will actually do by showing the dynamic behavior between objects in the system. Usually, behavior can be represented as hierarchies or networks of tasks, functions and processes. Behavior is evaluated via attributes of performance. The most common behavior diagrams include the use case diagram, activity diagram, and sequence diagram. All three are addressed in this paper using higraphs. Like requirements and structure, system behaviors can be easily represented in higraphs with advantage over current modeling tools.

Use Case Diagram

As seen earlier in this paper, the use case diagram provides a high-level description of the system functionality. Use cases describe behavior in terms of the high level functionality and uses of a system, that are further specified in the other behavioral diagrams referred to above. Use cases are often considered the most critical diagram and therefore often one of the first diagrams used to model a system. It is for this reason that use case diagrams and their higraph equivalent were covered in depth earlier in this paper.

Activity Diagram

The activity diagram represents the step-by-step flow of data and control between activities within a system. Activity diagrams detail a use case's activities (nodes), transitions (arcs), decision elements, and parallel behaviors. Figure 33 models a course registration use case (excluding update grades) using a UML activity diagram.

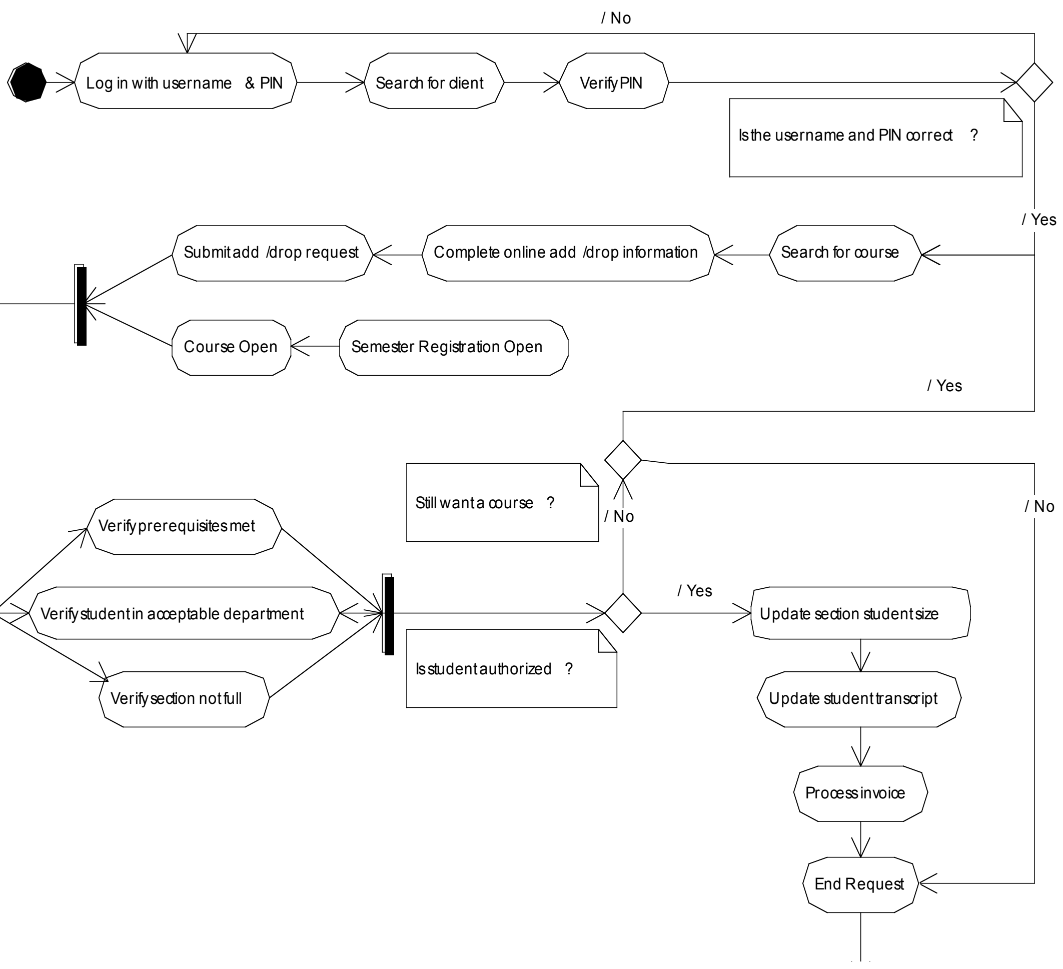
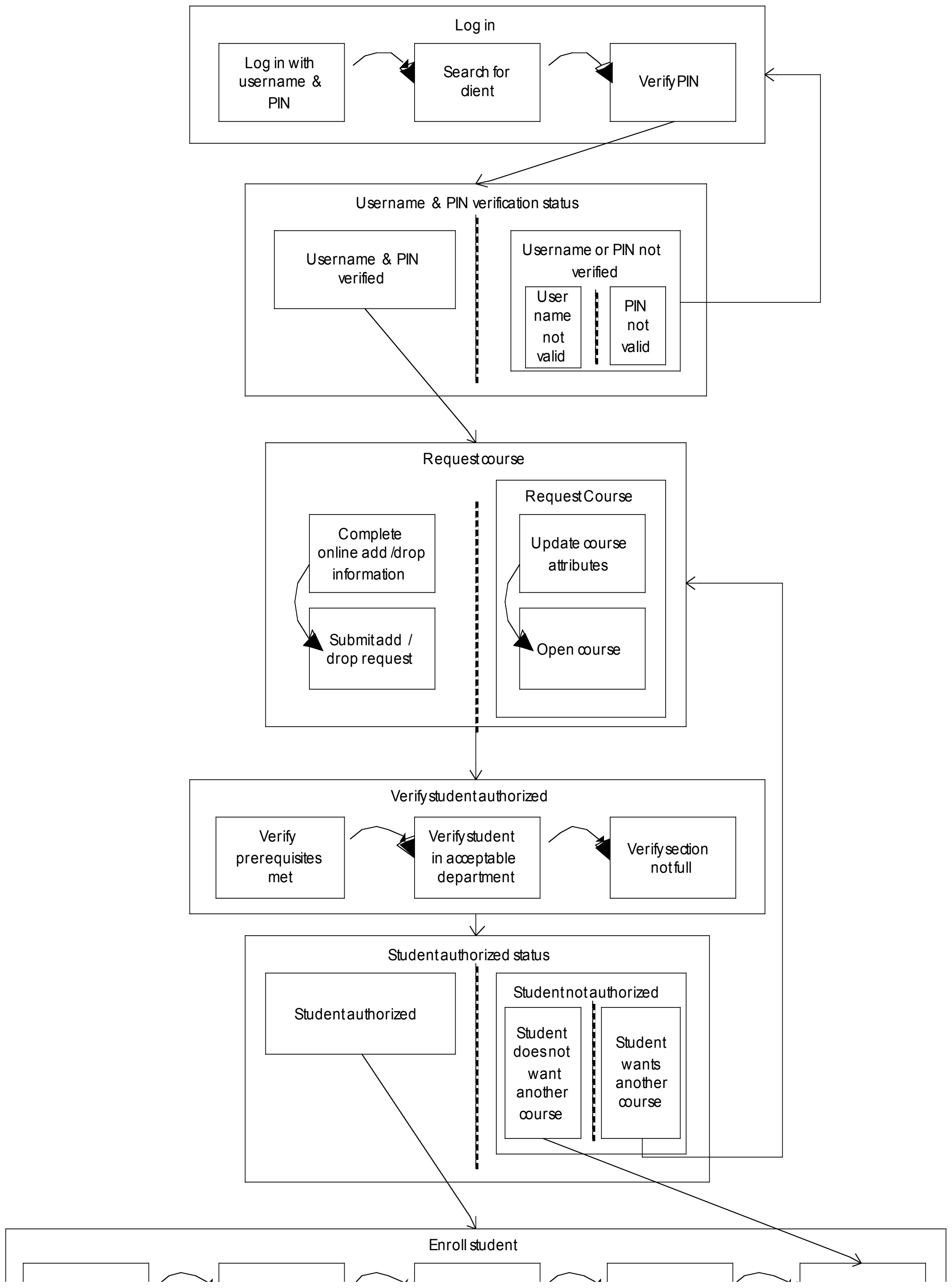


Figure 33: UML Activity Diagram representing a course registration use case

The transition between activity diagram and activity higraph is particularly natural Activity diagram's nodes and arcs can be equally represented in a higraph as blobs and edges. Decision elements and parallel behaviors would be supported by orthogonally divided activities. Orthogonal lines for decision elements are labeled as "or", and orthogonal lines for parallel/concurrent behaviors are labeled as "and". Like all other higraphs, activity higraphs use hierarchies to represent different levels of behavior.

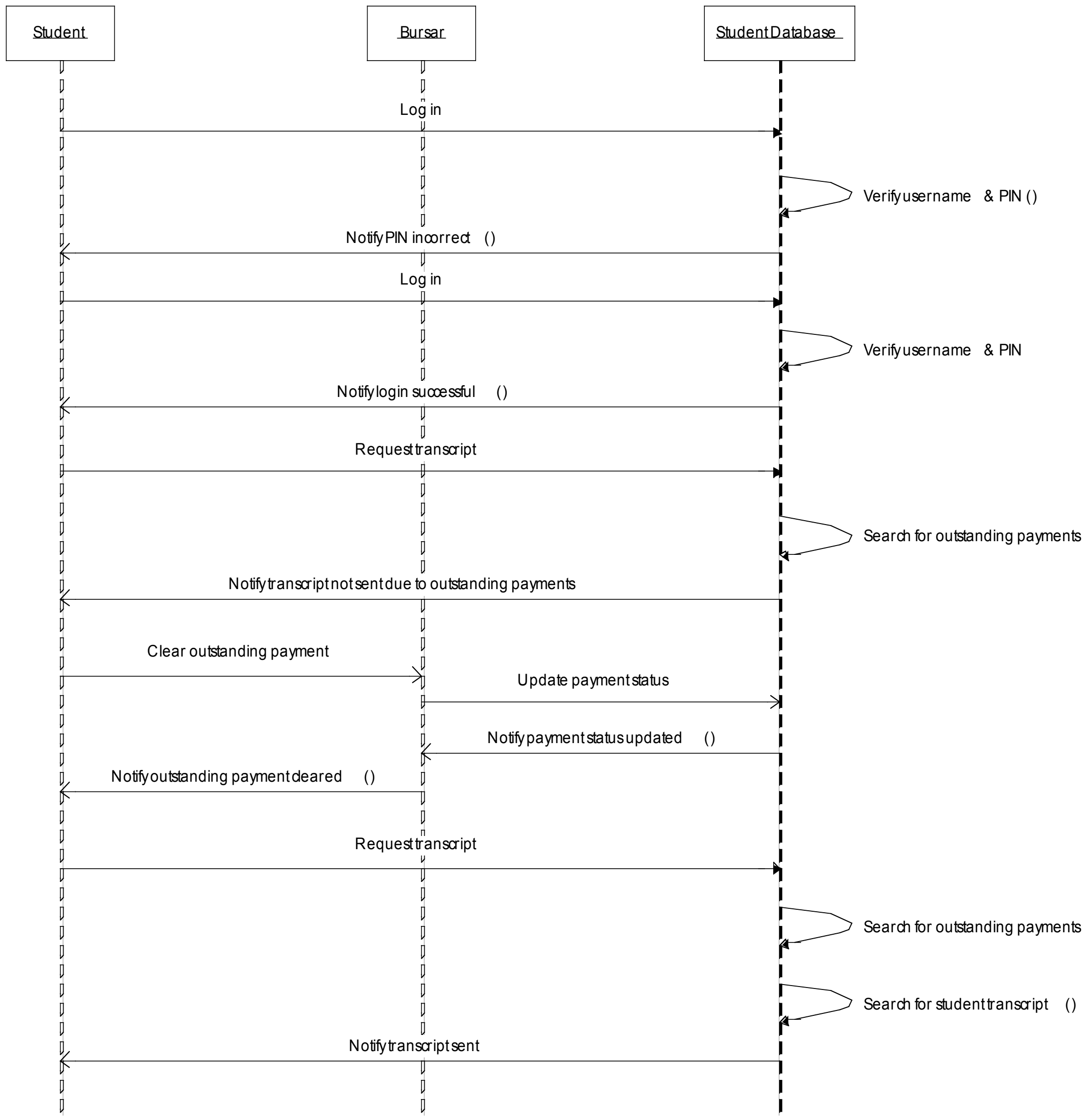


Log in
Log in with username & PIN
Search for client
Verify PIN
Username & PIN verification status
Username & PIN verified
Username or PIN not verified
User name not valid
PIN not valid
Request course
Complete online add /drop information
Submit add / drop request
Request Course
Update course attributes
Open course
Verify student authorized
Verify prerequisites met
Verify student in acceptable department
Verify section not full
Student authorized status
Student authorized
Student not authorized
Student does not want another course
Student wants another course
Enroll student

Figure 34: Activity Higraph representing a course registration use case

Sequence Diagram

The UML sequence diagram represents the interaction between objects of a system over time. These interactions are know as messages which when combined to create a sequence describe the behavior of the system, subsystem, activity, or use case. A sequence diagram detailing the University Registration System's request transcript use case is provide in Figure 35. As shown Figure 35, arrowed lines from one object, object A, to a second object, object B, are used to illustrate an outgoing message for object A and an incoming message for object B. A sequence is modeled by connecting the messages to the border of the objects



Student
Bursar
Student Database
Log in
Verify username & PIN ()
Notify PIN incorrect ()
Log in
Verify username & PIN
Notify login successful ()
Request transcript
Search for outstanding payments
Notify transcript not sent due to outstanding payments
Clear outstanding payment
Update payment status
Notify payment status updated ()
Notify outstanding payment cleared ()
Request transcript
Search for outstanding payments
Search for student transcript ()
Notify transcript sent

Figure 35: UML Sequence Diagram representing the request transcript use case

A sequence diagram is easily transformed into a sequence higraph as shown in Figure 36. In sequence higraphs, a message is represented by a combination of edges and (time) blobs. The originating edge of a message begins at the sending object (ie. student, bursar, student database), then passes through the message time blob (possibly with an attribute counting time), and ends with an originating message edge at the receiving object.

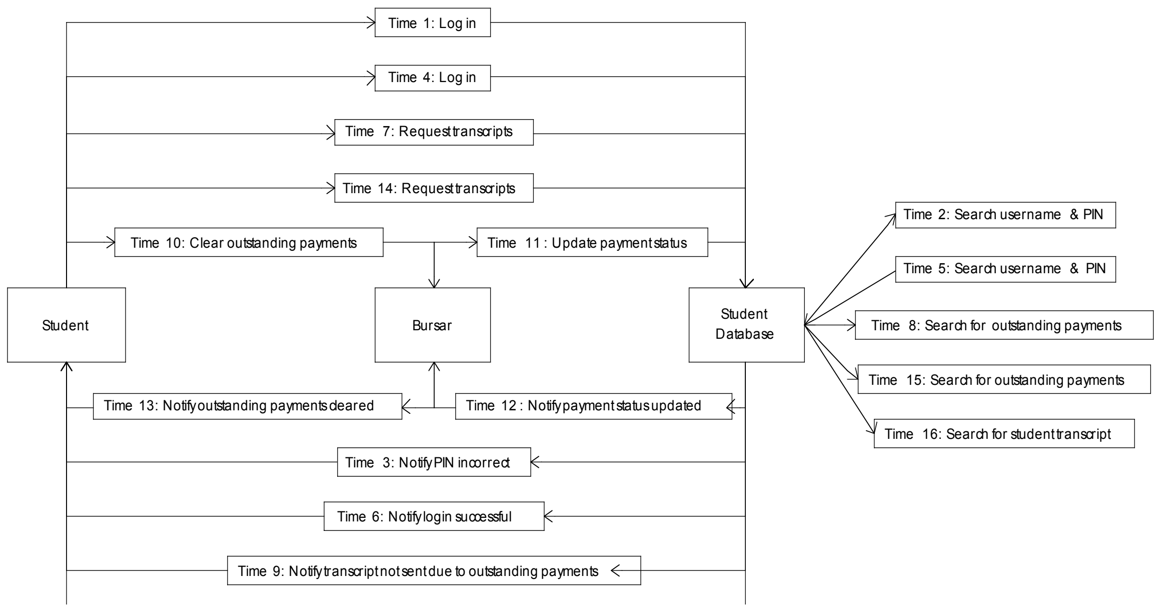

Time 1: Log in
Time 4: Log in
Time 7: Request transcripts
Time 14: Request transcripts
Time 10: Clear outstanding payments
Time 11 : Update payment status
Student
Bursar
Student Database
Time 2: Search username & PIN
Time 5: Search username & PIN
Time 8: Search for outstanding payments
Time 15: Search for outstanding payments
Time 16: Search for student transcript
Time 13: Notify outstanding payments cleared
Time 12 : Notify payment status updated
Time 3: Notify PIN incorrect
Time 6: Notify login successful
Time 9: Notify transcript not sent due to outstanding payments

Figure 36: Sequence Higraph representing the request transcript use case

Conclusion

Based on existing uses for higraphs, theories presented in Fogarty's thesis, and examples in this paper, higraphs have the potential to be a useful tool for complete system modeling. They are detailed and logical to follow, while filling the traceability gap often identified with existing modeling tools. Higraph's underlying properties create and display general connectivity and organizational relationships among components (behavior and structure) and requirements. Due to these characteristics they are able to accurately and comprehensively classify components within a system. The example presented demonstrates how higraphs can clearly represent all required information and formally show all relationships in the model through hierarchies, edges, and orthogonalities. Higraph's properties allow for a smart model that is visually logical, can be queried for custom sets, and has the potential to be easily adapted for software use. When presented to the systems engineer, higraphs substantially aid in engineering decisions. Although this paper graphically illustrates how higraphs can be used as a system modeling tool, it is important to note that in no way can it be used as an independent modeling language. It is a tool to complement existing languages such as UML and SysML. The incorporation of these techniques can overcome some of UML and SysML's existing criticisms. It therefore seems reasonable that higraph representations can compliment, and perhaps even co-exist, with UML and SysML representations of systems [1]. Higraphs offer 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 [1].

Bibliography

1. Fogarty, Kevin. System Modeling and Traceability. Diss. The Institute for Systems Research, Univ. of Maryland At College Park, 2006.

2. *Friedenthal, Stanford INCOSE 2004 Symposium Paper "*SysML Overview; *Extending UML to Support a Systems Modeling Language"*

3. Grossman, Ornit. Harel, David. .On the Algorithmics of Higraphs.. Technical Report CS97-15, The Weizmann Institute of Science, Rehovot, Israel, 1997

4. Harel, David. .On Visual Formalisms.. Communications of the ACM 31 (1988): 514-530.

5. "News" Headway Software, Inc. 11 Jul. 2007 http://headwaysoftware.com/about/customers.php

6. "Unified Modeling Language." Wikipedia. 23 Apr. 2007. 24 Apr. 2007 <http://en.wikipedia.org/wiki/Unified_Modeling_Language>.