Journal of Information Technology Impact
Vol. 3, No. 1, pp. 25-44, 2003

The Impact of UML Class Diagrams on Knowledge Modeling, Discovery and Presentations

Bogdan D. Czejdo [1]
Loyola University
Louisiana, USA

Rudolph L. Mappus IV [2]
Mapsoft Consulting
Texas, USA

Kenneth Messa [3]
Loyola University
Louisiana, USA

Abstract

Typically, the Unified Modeling Language (UML) is used for visualizing, specifying, constructing, and documenting the artifacts of software-intensive systems. However, there have been some projects showing the usefulness of UML modeling of systems in other areas. In this paper we discuss how to model knowledge from an example subject area and how to convert this model into a well-structured UML graph. UML models can be built to gain a deeper understanding of a subject area, to guide knowledge discovery, to prepare better presentations about the subject (including Web presentations), or to support the learning process. We concentrate our discussion on the impact of UML diagrams on improving learning and, specifically, on new generation Computer Assisted Instruction (CAI) tools that are based on UML diagrams.

Keywords: Unified Modeling Language, Computer Assisted Instruction, Learning.

Introduction

Typically, the Unified Modeling Language (UML) is used for visualizing, specifying, constructing, and documenting the artifacts of software-intensive systems (Booch et al., 1999; Rumbaugh, et al., 1999). There have been some projects, however, that show the usefulness of UML modeling of systems not necessarily involving software (Baszun, Czejdo & Miescicki, 1996). James Rumbaugh (1993), one of the authors of the UML, showed examples of how to model the United States Constitution. Generally, we can use UML to capture classifications that constitute the basic knowledge about variety of subjects. These classifications can be based on a subclass hierarchy, an aggregation hierarchy, named association relationships, or any combination of these. Such classifications can be used to develop a deeper understanding of the subject area, to guide in discovery of additional knowledge, to prepare better presentations about the subject, including Web presentations, or to support the learning process.

In this paper we discuss how to create UML diagrams by capturing knowledge from a sample domain specified in a natural language. We show how to transform a UML diagram in any subject area into a well-structured UML diagram. This transformation is a guided knowledge discovery process in which new information is requested and added to the diagram in a controlled manner. Here, we concentrate on transformation techniques related with aggregations. These

techniques reflect the basic observation that some parts of an entity and the entities themselves may have the same properties and responsibilities. We will refer to this situation as "subsumption". The main difference between subsumption and inheritance is that subsumption needs to be determined for each aggregation relationship, whereas inheritance is generally automatic (Booch et al., 1999). Therefore a significant part of the process of identifying missing knowledge in UML diagrams involves subsumptions in aggregations.

Next we discuss how a UML diagram can serve as a basis for a structured presentation. We also discuss how UML diagrams and their transformations can be used in the educational process. We propose a new generation of Computer-Aided Instruction (CAI) systems using UML modeling so that not only the human instructor, but also the CAI system, can use our methodology to guide a student's knowledge discovery.

UML Diagrams for Knowledge Modeling

Most of a subject's knowledge is in the form of a natural language. There are many materials and visualizations that can help to understand the subject matter, but typically these do not provide a complete framework for general ideas presented in a natural language. One of the difficulties in knowledge extraction is related to the ambiguity of natural language.

In this section we will concentrate on creation of initial UML diagrams for knowledge modeling.

The UML, though created and used for software design, is not specific to software design. This kind of modeling is *object oriented*, meaning that whatever system is being modeled, its components become abstract objects that have some properties (also called *attributes*) and functions (also referred to as *responsibilities*). A *class* is a collection of these abstract objects. Generally, there are several types of UML diagrams that can be useful for knowledge modeling, including class diagrams and state diagrams.

Class diagrams contain classes and relationships. Classes can be described by their name, properties, and functions, and they are graphically represented as boxes. Lines or arrows are then drawn between classes to describe their relationships, the most common of which are aggregation, generalization, and named association. Typically, aggregation is treated as a special form of association (Booch et al., 1999), but since aggregations play an important role in this paper, for convenience, we will discuss aggregations separately from other associations, which are referred to here as *named associations*. These different relationships are indicated by the symbols shown in Figure 1.

In order for UML diagrams to capture knowledge from various subject areas, some additional semantics and constraints need to be added to better model natural language formulations. Since there are many occurrences of similar properties and functions, a labeling system needs to be created to clearly denote connections between properties in different classes and between functions in different classes.

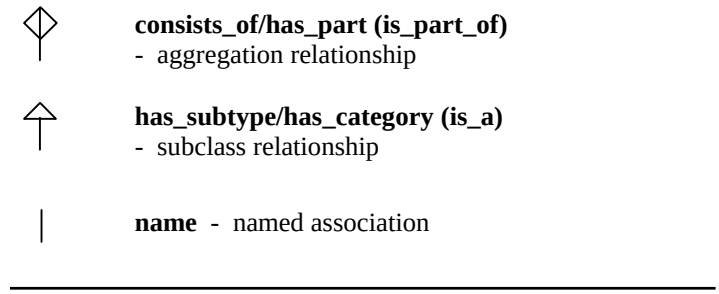

Figure 1. UML Relationships

As an example, let us consider the knowledge contained in the first chapter of a typical introductory text on computer literacy (Capron & Johnson, 2002). Let us assume that a reader, studying this textbook, recalls phrases from the discussion on computer systems and now wishes to model them in a UML diagram. First, the reader recognizes properties of the computer system in the statement: “Computer systems are characterized by speed and reliability.” The reader adds a new class to the diagram named "Computer System" and adds to it the two properties: "speed" and "reliability" and labels them as "A" and "B". The reader also realizes that the Computer System class has a function from the statement, “The main function of a computer system is to process data into information.” and labels it as "1". See Figure 2.

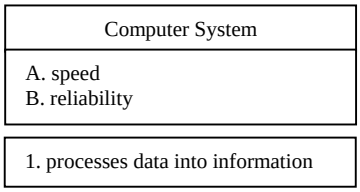

Figure 2. First step of a reader’s UML Model

Next, the reader uses the statement “Hardware is one component of any computer system,” to discover an aggregation relationship with the Computer System class. The reader adds the "Hardware" class and links it with the Computer System class using an aggregate relationship. The reader further realizes that the Hardware class has an aggregation relationship with the new class Input, which, in turn, also has an association with the classes "Data" and "Command", because of the statement: “One component of the hardware of a computer system is input. Input accepts data and commands and sends them for processing.” The reader adds the new classes and links them to the Hardware class. The resulting UML diagram is shown in Figure 3.

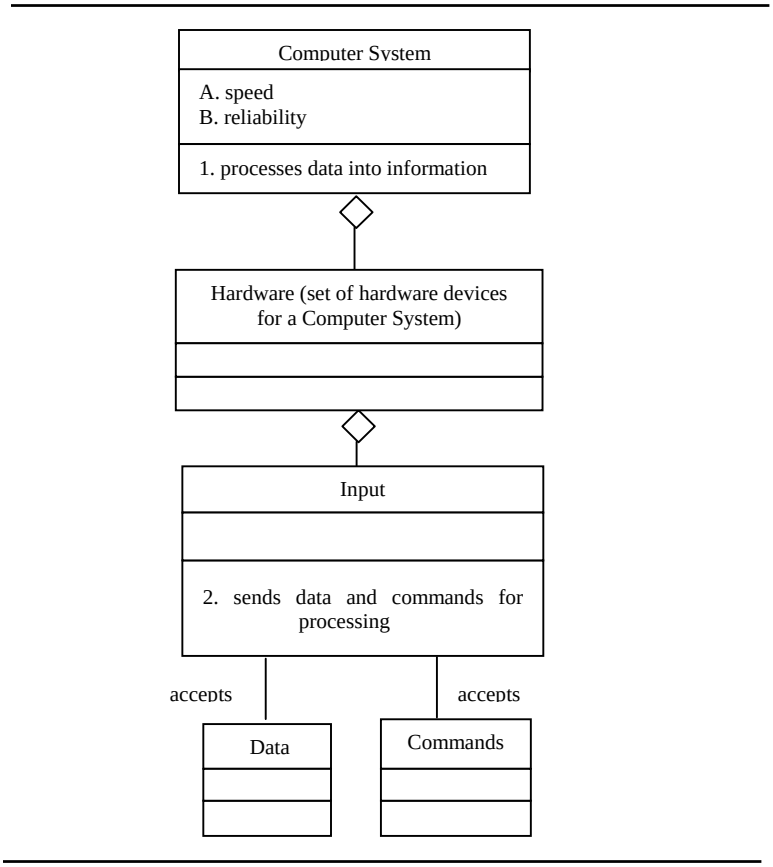

Figure 3. Second step of a reader's UML Model

With just a few more steps the reader can complete the diagram as shown in Figure 4. Notice that the names of the properties are chosen in such a way that they imply some values e.g. "speed".

Class diagrams present mainly static knowledge (a static view) of the subject area. If the subject area also contains a much information about processes or states of objects (an active view), it may in addition, be modeled by some other type of UML model, such as a UML state diagram. A state diagram has a collection of states that correspond to the phases of the modeled process. This type of diagram is often helpful when there are explicit or implicit references to states in the subject area, like steps or phases in a technological process. Since this type of UML modeling requires a different methodology of presentation and has different uses in the educational process, it will not be discussed further in this paper.

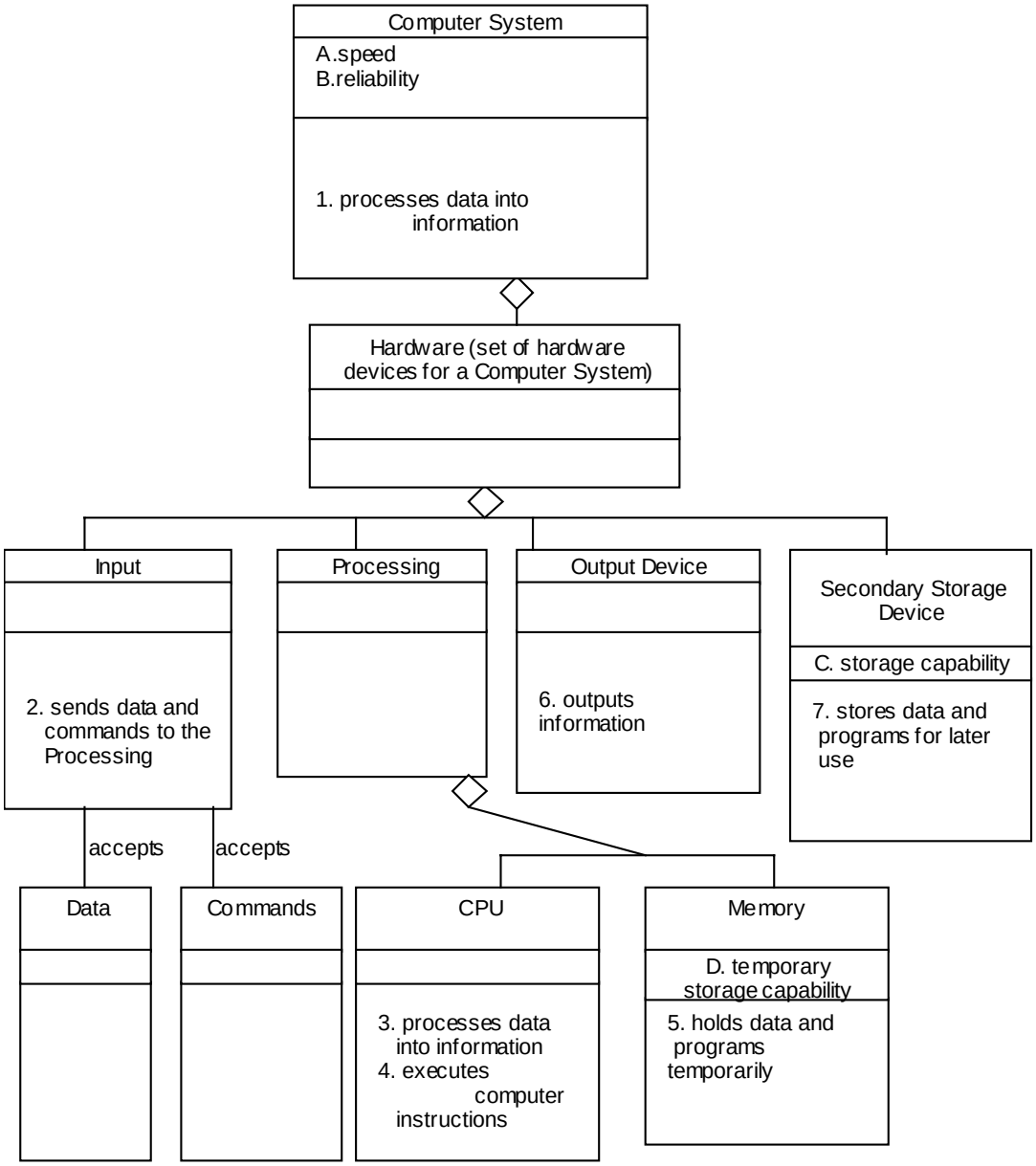

Figure 4. Third step of the reader's UML Model

Guided knowledge discovery using UML diagram

Initial UML diagrams capture the subject area's knowledge. However, missing knowledge can also be identified and discovered with a guided knowledge discovery process. By "guided knowledge discovery" we mean the controlled process in which a person would be guided by questions suggesting possible additions, deletions, or rewritings of properties, functions and relationships in the UML diagrams. An interesting aspect of this process is that a person can also use his/her own knowledge of the subject area to enhance or reinterpret the natural language formulations and to transform the UML diagram into a more complete structure.

We will classify UML transformations according to different techniques of knowledge discovery. For example, during a discussion on computer systems, a student may not be able to capture all of the information initially presented. The student will then need to discover new or missing knowledge and find missing references to this omitted knowledge. In this paper we concentrate on techniques related with aggregations. These techniques reflect the basic observation that some parts of an entity and the entities themselves may have the same properties and responsibilities. We refer to this situation as "subsumption". The first technique is related to considering subsumption and finding similarities between properties. The second technique is related to considering subsumption and finding similarities between functions/responsibilities. The third technique is related to considering subsumption and finding similarities between named associations. The fourth technique is related to converting properties and functions/responsibilities into named associations. The fifth technique is related to converting named associations into properties or functions/responsibilities. These techniques can be described by transformation rules for UML diagram modification. Therefore there are five groups of transformation rules – Property, Responsibility, Association, Refinement Expanding and Refinement Collapsing Rules. We describe each in turn.

First Group

The first group of rules primarily involves properties, so we refer to them as Property Rules. The Property Rules are based on the above-described concept of *subsumption*. Specifically,

- Parts of an entity need to be identified that have the same properties as the entities themselves and vice-versa.
- Since there might be more specific descriptions of a property on lower levels of a class diagram, property specialization should be allowed. By "property specialization" we mean a property that is derived from another property and has the same label, but which contains more specific information.
- In general, a property can include or overlap other properties. In the case of overlapping properties, decomposing a property into smaller components or combining properties into a single property may aid in creating a proper knowledge structure and improve the understanding of the subject.

Property Rule 1. Traverse the entire diagram from the top down to determine for each child (related by aggregation relationships) what properties are subsumed. Each property should be uniquely labeled, e.g. by a capital letter. While traversing the diagram, all existing properties

should be compared for similarity, especially similar concepts on different levels of the diagram. Rephrase the properties if possible to make them compatible.

For the diagram in Figure 4, the application of Property Rule 1 allows a student to consider if Hardware objects subsume both Computer System properties, "speed" and "reliability". By affirming this, a student actually discovers new knowledge about Hardware. This guided discovery is based on the fact that a student does not need to invent new properties, but only answer yes/no questions about hardware and its specific properties. Similarly a student can also conclude that the Processing and Secondary Storage classes have the property “speed” and that all hardware components have the property "reliable".

Property Rule 2. Traverse the whole diagram from the bottom up to determine for each parent (related by aggregation relationships) what properties can be propagated to the parent.

For the diagram in Figure 4, the application of Property Rule 2 allows a student to consider if “storage capability” of “Secondary Storage Device” is more general and can be applied to Hardware and Computer System. By affirming this, a student discovers new knowledge about hardware and computer systems. Similarly a student can also conclude that "temporary storage capability" in the Memory class does not propagate upwards.

Property Rule 3. Traverse the whole diagram comparing siblings' properties for similarity and, generally, any properties that have not been compared for similarity. Rephrase the properties if possible to make them compatible. Identify missing properties. Consider negated or complementary properties, as well.

For the diagram in Figure 4, the application of Property Rule 3 allows a student to consider if “storage capability” is similar to “temporary storage capability”. By adding "permanent" to "storage capability", a student can better differentiate between those two different properties.

Second Group

The second group of rules involves responsibilities/functions, so we refer to them as Responsibility Rules. The Responsibility Rules are also based on the concept of subsumption. Specifically,

- Parts of an entity need to be identified that have the same responsibilities/functions as the entities themselves and vice-versa.
- Since there might be more specific descriptions of a function on the lower level of a class diagram, function specialization should be allowed. By "function specialization" we mean that a function is derived from another function and has the same label, but contains more specific information.
- In general, a function can include or overlap other functions. In the case of overlapping functions, decomposing a function into more primitive functions or combining functions into a single function may aid in creating a proper knowledge structure and improve the understanding of the subject.

Responsibility Rule 1. Traverse the entire diagram from the top down to determine for each child (related by aggregation relationships) which responsibilities/functions are subsumed. Each responsibility/function should be uniquely labeled, e.g. with a number. While traversing the diagram, all existing responsibilities/functions should be compared for similarity, especially responsibilities/functions of classes on different levels of the diagram. Rephrase the responsibilities/functions, if possible, to make them compatible.

For the diagram in Figure 4, the application of Responsibility Rule 1 allows a student to consider if the Hardware and Processing classes subsume the "processes data into information" function defined in the Computer System class. By affirming this, a student actually includes new knowledge about the Hardware and Processing classes in the diagram. A student may also recognize that functions 1 and 3 are identical and function 3 should be relabeled as 1.

Responsibility Rule 2 Traverse the whole diagram from the bottom up to determine for each parent (related by aggregation relationships) which responsibilities/function**s** can be propagated to each parent.

For the diagram in Figure 4, the application of Responsibility Rule 2 allows a student to consider if the function defined in the Output Device class ("outputs information") is more general and can be applied to the Hardware and Computer System classes. By affirming this, a student discovers new knowledge about Hardware and Computer System. Similarly a student can also conclude that the "executes computer instructions" function defined in the CPU class also propagates upwards.

Responsibility Rule 3 Traverse the entire diagram again and compare all classes' responsibilities/functions for similarity and completeness. By "completeness" it is meant that missing functions should be identified. Functions should be rephrased, if possible, to make them compatible.

For the diagram in Figure 4, the application of Responsibility Rule 3 allows a student to consider the "sends data and commands to processing" function defined in the Input class. An examination of the Processing class shows no corresponding function for receiving data and commands sent by an Input object. By adding the corresponding function to the Processing class, namely "accepts data and commands", a student can realize the relationship between the input and processing objects.

Third Group

The third group of rules involves named associations, so they are called Association Rules. The Association Rules are also based on subsumption. Specifically,

- Parts of an entity need to be identified that can be connected to the same classes as the entities themselves and vice-versa.
- Since there might be more specific descriptions of a named association on the lower level of a relationship, specialization should be allowed. By "association

specialization" we mean that the given named association is derived from another named association and has the same label, but contains more specific information.

- In general, an named association can include or overlap other named associations. In the case of overlapping named associations, decomposing an named association into more specific named associations or combining associations into a single one aids in creating a proper knowledge structure and improves the understanding of the subject.

Association Rule 1. Each named association should be uniquely labeled, e.g. by a Roman numeral. Repeat the traversal of the entire diagram from the top down to determine for each child (related by an aggregation relationship), which named associations can be subsumed. While traversing the diagram, all existing relationships (especially named associations) should be compared for similarity.

For the diagram in Figure 4, the application of Association Rule 1 would result in labels "I" and "II" being added to the two named associations both named "accepts".

Association Rule 2. Traverse the whole diagram from the bottom up to determine for each parent (related by aggregation relationships) which named associations can be propagated to the parent.

For the diagram in Figure 4, the application of Association Rule 2 allows a student to consider the “accepts” association between the Data and Command classes and the Input class. The named associations can be propagated upward to the Hardware and Computer System classes. The student gains a greater understanding of the role of hardware and computer systems.

Association Rule 3. Traverse the entire diagram and compare all named associations for similarity. The label of the named association should be changed or the association should be renamed if necessary.

For the diagram in Figure 4, there are no examples of this rule.

Fourth Group

The fourth group of rules is related with converting the properties and functions in the diagram; hence they are called Property and Function Conversion Rules. These rules are based on the fact that:

- Sometimes properties of a class can be converted into a named association with an existing class or possibly with a currently non-existent class. If the second class is shown on the diagram, then typically it is good idea to transform the property into a named association. If the second class is not shown on the diagram, but is referred to in several properties or functions, then the addition of a new class to the diagram should be seriously considered.

- Very often a function can be converted into a named association with an existent or possibly with a currently non-existent class. If the second class is shown on the diagram, then typically it is good idea to transform the function into a named

association. If the second class is not shown on the diagram, but is referred to in many properties or functions, then the creation of a new class should be seriously considered.

Property and Function Conversion Rule 1. Consider converting properties into named associations if they refer to another class.

Property and Function Conversion Rule 2. Consider converting responsibilities/functions into named associations if they refer to another class.

For the diagram in Figure 4, the application of these rules allows the student to consider the replacement of the function "2. sends data and commands to processing" by the association "sends data and instructions to" between the Input and Processing classes. A student again gains a greater understanding of the cooperation of the input and processing objects.

Fifth Group

The fifth group of rules is related with converting the named associations in a diagram; hence we call them Association Conversion Rules. We apply these rules when a diagram contains knowledge that is too detailed in terms of named associations, the associations are hard to read, or the designer prefers a natural language description of properties and functions.

The rules are based on the fact that a named association can be converted into a property or a function of a class. Since the named association connects two or more classes, the problem is to choose to which class the property or function should be attached. Choosing the primary class is difficult to do automatically and practically needs to be done by a guided knowledge discovery process.

Association Conversion Rule 1. Named associations can be replaced with corresponding properties.

Association Conversion Rule 2. Named associations can be replaced with corresponding function or responsibilities.

For the diagram in Figure 4, the application of Association Conversion Rule 2 allows a student to consider the integration of the Data and Command classes and named associations with the Input class as a function of the Input class. This clarifies the diagram and adds a descriptive element to the Data and Command classes' named association with the Input class.

Applying these basic rules, adding multiplicities (whose description is omitted) and indicating subsumptions with "(S)" would result in the diagram shown in Figure 5a.

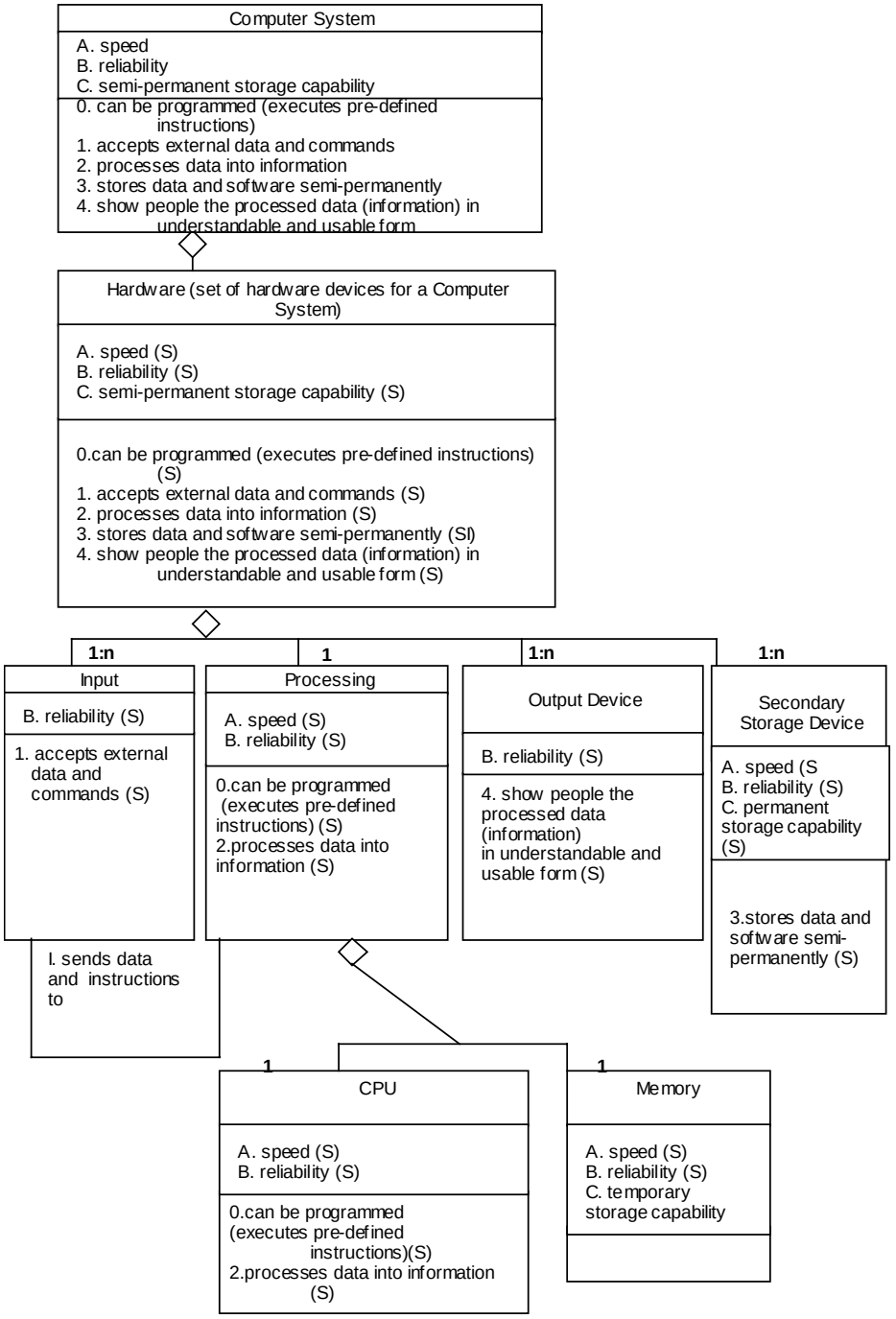

Figure 5a. Fourth step of a reader's UML Model

Knowledge Refinement for Presentations

The transformed UML diagram allows the capture of the subject area's knowledge in a well-organized manner. However, these diagrams might need to be further refined in order to be used efficiently for preparing a presentation. In this section we will concentrate on refinement of UML diagrams for the purpose of a presentation. We will discuss the guided refinement process in which a person would be guided to refine the diagram according to her/his needs. There are two types of presentation refinements. The first type is related with expanding the diagram, adding additional knowledge, and including additional classification levels. The second type is related with hiding some knowledge and removing some classification levels. These two types of refinements correspond to two types of refinement rules. The first group of refinement rules is called Refinement Expanding Rules and the second group is called Refinement Collapsing Rules. We discuss each in turn.

First Group

The Refinement Expanding Rules are based on the fact that:

- Sometimes classes with similar properties, functions or named associations can be grouped into categories.
- Sometimes the aggregation hierarchy or classification hierarchy can be made more precise by adding additional hierarchy levels.

Refinement Expanding Rule 1. Investigate whether classes can be classified into groups (categories) and, if so, insert new classes corresponding to the categories. The new category classes should be connected by subclass relationships to the existing classes.

For the diagram in Figure 4, the application of Refinement Expanding Rule 1 allows the presenter to consider the creation of two new classes: I/O Devices and Other Hardware.

Refinement Expanding Rule 2. Investigate whether additional layers can be added to the aggregation hierarchy and insert new classes. The new classes should be connected by aggregate relationships with the existing components.

Refinement Expanding Rule 3. Investigate whether a diagram contains a class whose objects can be classified based on either properties or functions. If so, create new subclasses.

For the diagram in Figure 4, the application of Refinement Expanding Rule 3 allows the student to consider the classification of a Secondary Storage Device into "very large storage devices", "large storage devices", and so forth.

Refinement Expanding Rule 4. Investigate whether a diagram contains a named association that can be expanded into two or more, named associations and a new class, which is connected by these associations.

Refinement Expanding Rule 5. Investigate whether a diagram contains a class that can be expanded into two or more classes and appropriate relationships between them.

Second Group

If the knowledge represented in a diagram is too detailed, we might want to hide some information. The second type of refinement rule, the Refinement Collapsing Rules, are responsible for that process. The rules are based on the fact that:

- Any property or function can be hidden
- Any named association can be hidden
- Any class can be hidden with two possible implications: the existing relationships can be removed or redefined
- Any subclasses of a class (with appropriate relationships) can be collapsed into a single class.
- The aggregation or classification hierarchy can be made less precise by removing some levels.

Refinement Collapsing Rule 1. Remove any property, function or named association. Adjust the labeling as necessary.

Refinement Collapsing Rule 2. Remove a class. Decide if the appropriate relationships need to be modified or removed.

For the diagram in Figure 4, the application of Refinement Collapsing Rule 2 allows a student to remove the Processing class from the aggregation hierarchy.

Refinement Collapsing Rule 3. Identify a collection of subclasses of a given class and collapse them into a single class.

Refinement Collapsing Rule 4. Remove one layer from a classification or aggregation hierarchy. Redirect appropriate relationships.

Applying these rules would result in the diagram shown in Figure 5b. The diagram shown in Figure 5b can be more appropriate for presentation for variety of reasons. We have to take into account the audience's knowledge and understanding in order to make our presentation successful. For example, the listeners can be accustomed to a certain classification that is used in their discipline. It is important to adjust the diagram to their needs. Our transformation rules allow for such an adjustment.

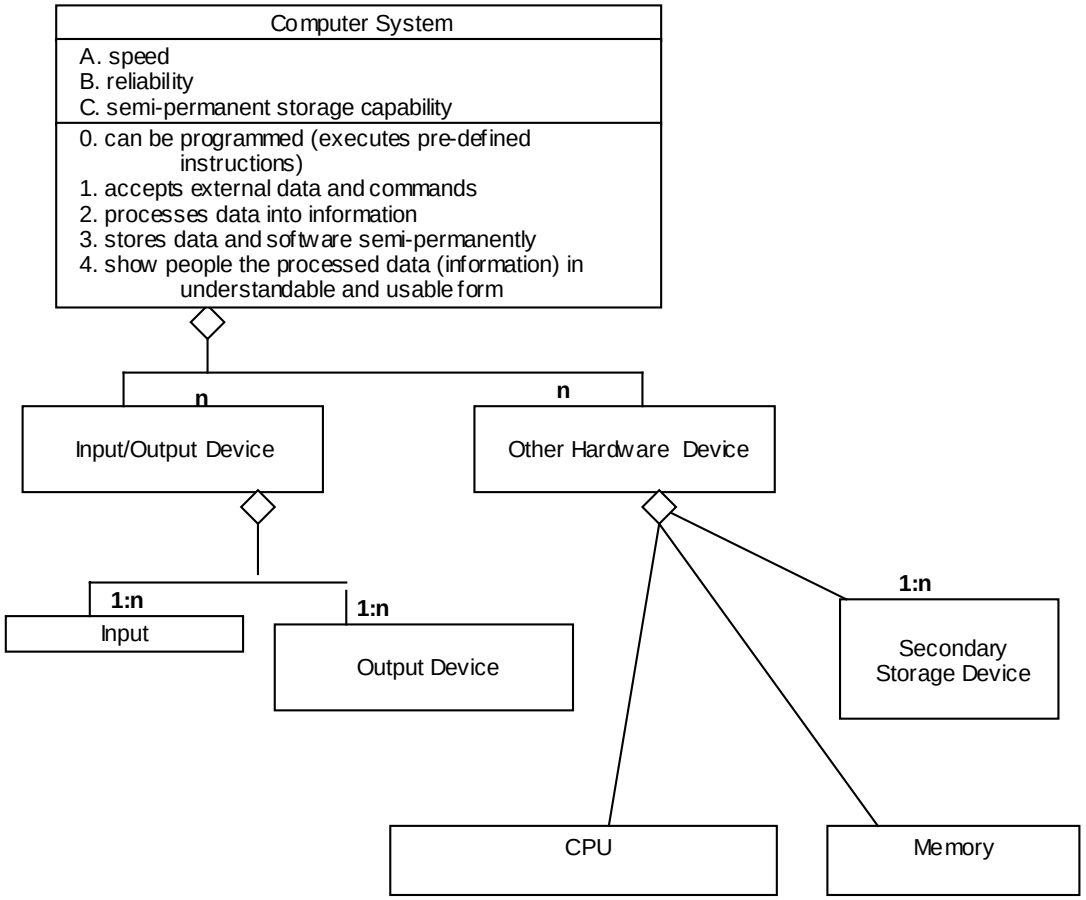

Figure 5b. Presentation UML Model

Computer Assisted Instruction and UML

The methodology described in the previous sections can be the basis for computer-assisted instruction (CAI) of knowledge modeling. CAI can support learning activities in various ways. Generally CAI systems allow for a spectrum of possibilities ranging from a minimal intrusion of the CAI system that guides through the possibilities and checks for significant mistakes to a rigid control of student activities that always shows “the best way". In our case, all possibilities on this spectrum can be applied. Let us discuss first the least intrusive mode of operation of our CAI system. In this mode the CAI system would accept the initial diagram, for example the diagram in Figure 4. A student would be allowed to apply any of the transformation rules. In each step the system would check significant constraints. These constraints are as follows:

a. No property label (a letter) can appear twice for different properties
b. No function label (a number) can appear twice for different functions
c. No named association label (a Roman numeral) can appear twice for different associations
d. A property identified as subsumed should have another property occurrence on the higher level
e. A function identified as subsumed should have another function occurrence on the higher level
f. A named association identified as subsumed should have another relationship occurrence on the higher level

Additionally, the CAI system can check if the UML diagram is well-formed. By "well-formed" we mean that all possible transformations had been considered. The architecture of such system is shown in Figure 6.

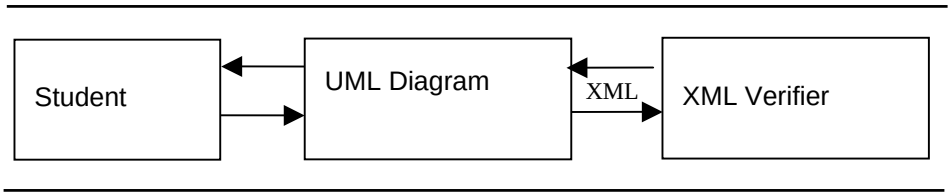

Figure 6. System architecture for less intrusive CAI system

A student uses the UML Diagram Tool to enter the initial UML diagram and to perform transformations for that diagram. The UML Diagram Tool generates an XML representation of the UML diagram. The XML data are sent to the XML Verifier (XML validating parser) that checks the above-described constraints according to the meta-data shown in Figure 7.

The property, SFlag, denotes whether the property or function is subsumed. It has one of four values: "new", "subsumed ", "refined", or "none". The attributes PFlag and FFlag denote whether the student considered it for comparison. It has two values "compared" and "not compared".

There are several states of UML transformations. The first state, called “initial”, corresponds to the situation when an initial UML diagram is created. When any transformation is specified, the state changes to “transformed”. When all possible transformations have been considered, the state changes to “well-formed”.

Let us discuss next a more intrusive mode of operation of our CAI system, when the instructor wants a student to follow her/his solutions. In this mode the CAI system would present to a student an initial diagram, for example the diagram in Figure 4. The student would be allowed to apply some of the transformation rules as indicated previously by the instructor. Even though the CAI system requires the student to reach the specific well-formed UML diagram, the order of application of the transformation rules is not imposed. In this way the student is allowed to discover new knowledge components that are tailored to his/her learning style.

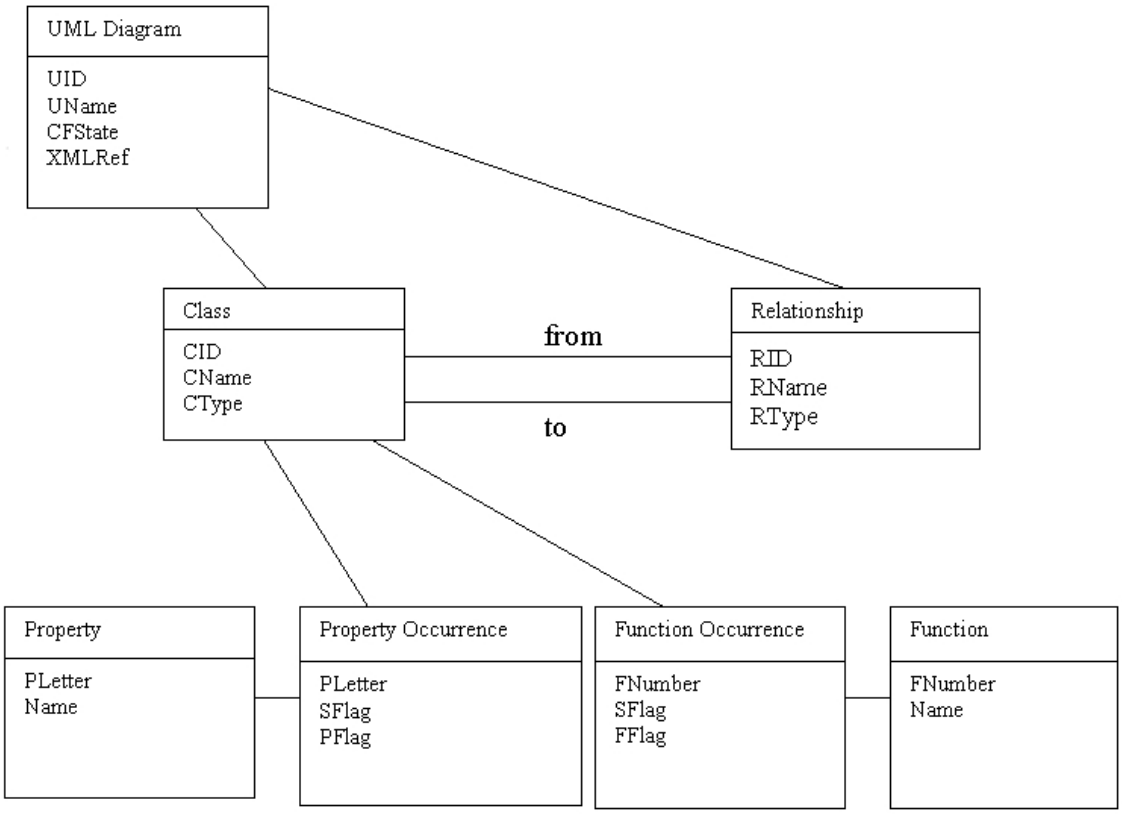

Figure 7. Meta-data for UML diagram

For example, he/she can choose the easiest questions first or can pursue the learning process in some order. In each step the system would check not only significant constraints, but also if the rules that the student applied belong to the set of answer rules previously stored by the instructor.

The architecture of such system is shown in Figures 8a and 8b.

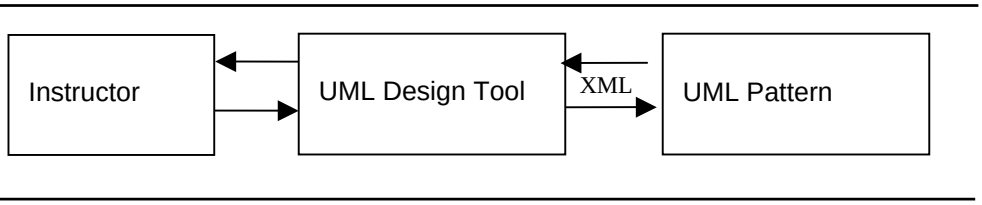

Figure 8a. System architecture for instructor preparing both initial and well-formed UML diagram

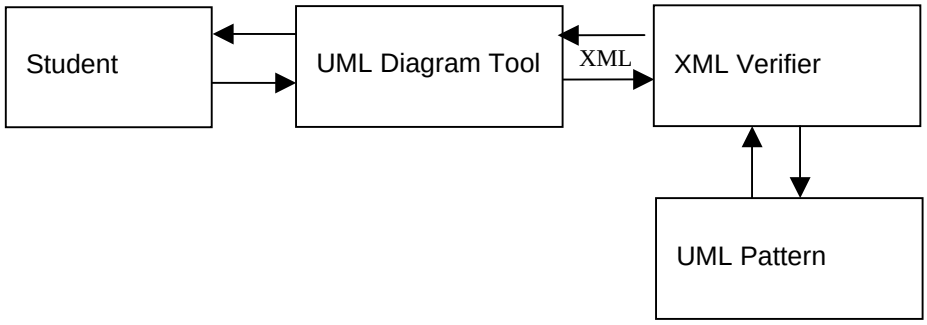

Figure 8b. System architecture for student performing guided transformations of UML diagram

In Figure 8a, the instructor uses the UML Design Tool to enter the initial UML diagram and to perform transformations for that diagram. The UML Diagram Tool generates an XML representation of the UML diagram. The XML data are sent to the UML Pattern module. The UML Pattern module keeps not only initial UML diagram, but also the final well-formed UML diagram. Then, as shown in Figure 8b, the student uses the UML Diagram Tool to perform transformations on the diagram. The UML Diagram Tool generates an XML representation of the UML diagram. The XML Verifier compares the student's UML diagram with the instructor's UML diagram stored in UML Pattern module. The student transformation is allowed, if it does not contradict the instructor's well-formed diagram.

Let us discuss next the most intrusive mode of operation of our CAI system where the instructor wants a student to follow not only her/his solutions, but also the steps on how to achieve them. This mode would typically be used by a student to observe the “best solution” set forth by the instructor. In this situation the CAI system would also display the initial diagram, for example the diagram in Figure 4 and all the transformations leading to the diagram in Figure 5. Alternatively the student would be allowed to strictly apply in a certain order the transformation rules as previously indicated by the instructor.

The architecture of such system is shown in Figures 9a and 9b.

Figure 9a. System architecture for instructor preparing ordered transformations of UML diagram system

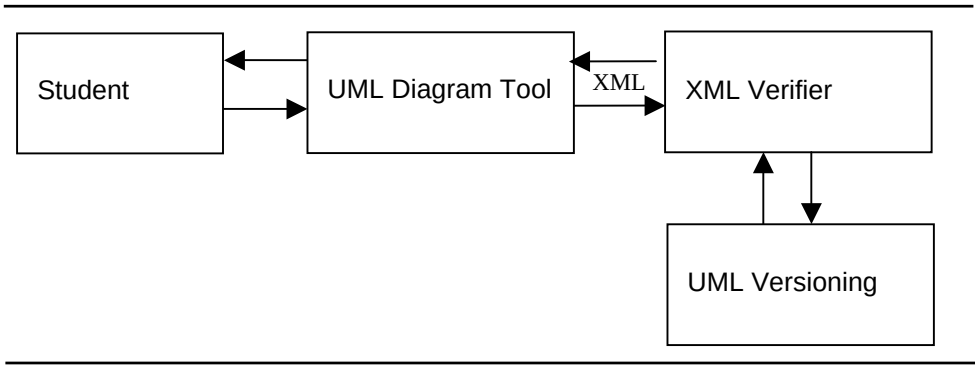

Figure 9b. System architecture for tutoring students.

In Figure 9a, the instructor uses the UML Design Tool to enter an initial UML diagram and to perform transformations for that diagram. The UML Diagram Tool generates an XML representation of the UML diagram. The XML data are sent to UML Versioning module that keeps not only initial UML diagram, but also the sequence of transformations using versioning techniques described in the literature (Beech & Mahbod, 1999; Katz, 1990). Then, as shown In Figure 9b, a student uses the UML Diagram Tool to observe transformations for that diagram.

Summary

In this paper we showed how to use the UML to capture basic knowledge about a subject area. This basic knowledge includes objects and their classes, properties and functions of classes and relationships between them. Relationships can be a good basis for classifications; that is, classifications can be based on a subclass hierarchy, an aggregation hierarchy, named associations, or any combination of these. Here we concentrated on aggregation relationships since their analysis is necessary to develop a deeper understanding of the subject area, and can serve as a basis for guiding discovery of additional knowledge. In this paper we also addressed the issues of preparation of presentations about the subject based on UML diagram, and support the learning process by using UML diagrams. In particular, the learning process can be significantly enhanced by considering UML diagram transformations that guide the students' knowledge discovery.

References

Baszun M., Czejdo B., Miescicki J. (1996). Modeling of Interactive Engineering Design. In *Proceedings of ESDA'96*. Montpellier, France.

Beech, D., Mahbod, B. (1999). Generalized Version Control in an Object-Oriented Database. In *Proceedings of the IEEE Data Engineering Conference*. Los Angeles: IEEE.

Booch, G., Rumbaugh, J. & Jacobson, I. (1999). *The Unified Modeling Language User Guide.* Addison Wesley.

Capron H. & Johnson J. (2002). *Computers Tools for an Information Age. Edition 7.* Prentice Hall.

Katz, R. (1990). Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys, 22* (4), 375-408.

Rumbaugh, J., Jacobson, I. & Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Addison Wesley.

Rumbaugh, J. (1993, January). Objects in the Constitution: enterprise modeling. *Journal of Object Oriented Programming.*

---

[1] Dr. Bogdan Czejdo is a professor at Loyola University New Orleans. He can be reached at: Department of Mathematics and Computer Science, Campus Box 35, Loyola University New Orleans, 6363 St. Charles Ave., LA 70118, USA. E-mail: czejdo@loyno.edu, Phone: (504) 865-3340, Fax: (504) 865-2051.

[2] Mr. Rudolph L. Mappus IV is president of Mapsoft Consulting and a doctoral student at the Georgia Institute of Technology. He can be reached at: Mapsoft Consulting, Dallas, TX 75238, USA. E-mail: cmappus@mapsoft.net.

[3] Dr. Kenneth Messa is a professor at Loyola University New Orleans. He can be reached at: Department of Mathematics and Computer Science, Campus Box 35, Loyola University New Orleans, 6363 St. Charles Ave., LA 70118, USA. E-mail: messa@loyno.edu, Phone: (504) 865-3135, Fax: (504) 865-2051.