UML (Unified Modeling Language): Standard Language for Software Architecture Development

2 authors:

Harpreet Kaur
Punjabi University, Patiala
7 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Pardeep Singh
National Institute of Technology
21 PUBLICATIONS **62** CITATIONS
SEE PROFILE

2009 International Symposium on Computing, Communication, and Control (ISCCC 2009)
Proc .of CSIT vol.1 (2011) © (2011) IACSIT Press, Singapore

UML (Unified Modeling Language): Standard Language for Software Architecture Development

Harpreet Kaur [1] and Pardeep Singh [2 +]

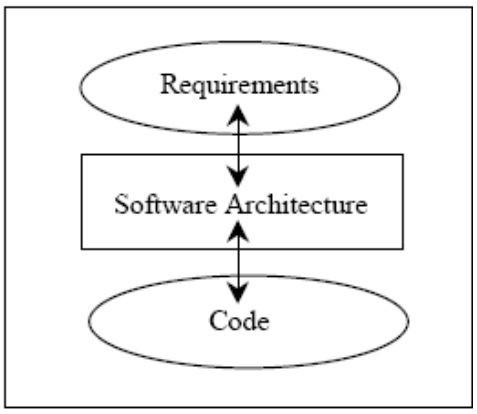
[1,2]Lecturer: [1]Thapar University, Patiala (India), [2]NIT Hamirpur (India)

Abstract. A solid architectural vision is a key discriminator in the success or failure of a software project. This paper discusses how to describe architecture through a set of design viewpoints and how to express these views in the UML. In this paper we mainly focused on the standardization of UML for software development process with the discussion of benefits of UML. Object-oriented analysis (OOA) is concerned with developing software engineering requirements and specifications that expressed as a system's object model (which is composed of a population of interacting objects), as opposed to the traditional data or functional views of systems. The goal of this paper is to combine the respective strengths of powerful, specialized (architectural-based) modeling approach (UML) with a widely used general (design-based) approach. Also some weaknesses have been discussed so that existing modeling can be optimized for better architecture development.

1. Introduction

The **software architecture** of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships between them. Software architecture typically plays a key role as a bridge between requirements and implementation. *An architecture is the **set of significant decisions** about the organization of a software system, the selection of **structural elements** and their interfaces by which the system is composed, together with their **behavior** as specified in the collaborations among those elements, the **composition** of these elements into progressively larger subsystems, and the **architectural style** that guides this organization -- these elements and their interfaces, their collaborations, and their composition* [1].

By providing an abstract description of a system, the architecture exposes certain properties, while hiding others. Ideally this representation provides an intellectually tractable guide to the overall system, permits designers to reason about the ability of a system to satisfy certain requirements, and suggests a blueprint for system construction and composition. All the above themes are discussed below with role of software architecture in software development [1].



+ Corresponding author.
E-mail address: (harpreet_khasria@rediffmail.com, p_singh79@rediffmail.com).

Fig 1: Software Architecture

2. Role of Software Architecture

While there are numerous definitions of software architecture, at the core of all of them is the notion that the

architecture of a system describes its gross structure. This structure illuminates the top-level design decisions, including things such as how the system is composed of interacting parts, where the main pathways of interaction, and what are the key properties of the parts. Additionally, an architectural description includes sufficient information to allow high-level analysis and critical appraisal [6]. Software architecture can play an important role in following aspects of software development and helps to define the following:

2.1. Software Structure is defined with Architecture

Software architecture is loosely defined as the organizational structure of a software system including components, connections, constraints, and rationale. Components can be small pieces of code, such as modules, or larger chunks, such a stand-alone program like database management systems. Connections in architecture are abstractions for how components interact in a system, e.g., procedure calls, pipes, and remote procedure calls. Architecture has various constraints and rationales associated with it, including the constraints on component selection and the rationale for choosing a specific component in a given situation. Figure 2 shows a UML class diagram containing some structural represents student-course-enrollment processing system. Here we see three classes – Course, Student and Professor. All the three classes related to each other with the help of **multi-dependency** relationship defined in UML, various attributes of classes are shown in the respective boxes with each class.

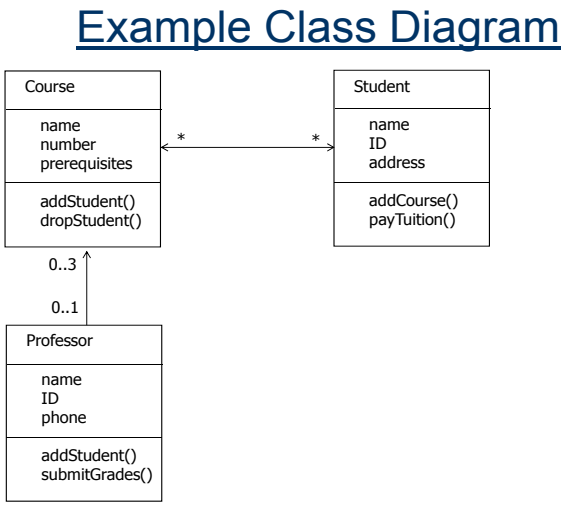


Fig 2: Relationships

2.2. Behavior is characterized with Architecture

The basic idea of architecture definition is to design software structure and object interaction (Behavior) before the detailed design phase. Behavior of software is really what our software does after it rolls out into the production environment. As well as defining structural elements, architecture defines the behavior of software system by defining interactions between structural elements.

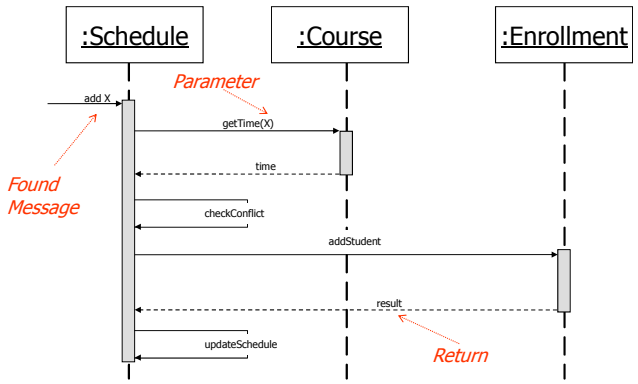


Fig 3: Sequence Diagram

Figure 3 shows a UML sequence diagram showing a number of interactions that, together, allow the system to support the creation of an course-enrollment for a student. Here we see three object interactions. First, a Schedule object interacts with the Course object to have time for particular course. The course instance then interacts back with Schedule object to check any conflict. Then Schedule instance get interacted with Enrollment object. Then depending upon the updated information particular students gets enrolled for specific course.

2.3. Focuses on Significant Elements

While architecture defines structure and behavior, it is not concerned with defining *all* of the structure and *all* of the behavior. It is only concerned with those elements that are deemed to be significant. Significant elements are those that have a long and lasting effect, such as the major structural elements, those elements associated with essential behavior, and those elements that address significant qualities such as reliability and scalability. In general, the architecture is not concerned with the fine-grained details of these elements. The architecture embodies information about how the elements relate to each other. This means that architecture specifically *omits* certain information about elements that does not pertain to their interaction. Thus, an architecture is foremost an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Since architecture focuses on significant elements only, it provides us with a particular perspective of the system under consideration -- the perspective that is most relevant to the architect. In this sense, architecture is an abstraction of the system that helps an architect manages complexity.

2.4. Architecture Balances Stakeholder Needs

Architecture is created to ultimately address a set of stakeholder needs. However, it is often not possible to meet all of the needs expressed. For example, a stakeholder may ask for some functionality within a specified timeframe, but these two needs (functionality and timeframe) are mutually exclusive. Either the scope can be reduced in order to meet the schedule or all of the functionality can be provided within an extended timeframe. Similarly, different stakeholders may express conflicting needs and, again, an appropriate balance must be achieved. Making tradeoffs is therefore an essential aspect of the architecting process, and negotiation, an essential characteristic of the architect. Another challenge for the architect is that the stakeholders are not only concerned that the system provides the required functionality. Many of the concerns listed are nonfunctional in nature in that they do not contribute to the functionality of the system (e.g., the concerns regarding costs and scheduling). Such concerns nevertheless represent system qualities or constraints. Nonfunctional requirements are quite often the most significant requirements as far as an architect is concerned.

2.5. An Architecture Influences Team Structure

Architecture defines coherent groupings of related elements that address a given set of concerns. For example, architecture for an order processing system may have defined groupings of elements for order entry, account management, customer management, and fulfillment, integrations with external systems, persistency, and security. Each of these groupings may require different skill sets. It therefore makes perfect sense to align software development team structures with the architecture once it has been defined. However, it is often the case that the architecture is influenced by the initial team structure and not vice versa. This is a pitfall that is best avoided, since the result is typically a less-than-ideal architecture. In practice, we often unintentionally create architectures that reflect the organization creating the architecture.

3. UML: Introduction

Several languages for describing software architectures have been devised, but no consensus has yet been reached on which symbol-set and view-system should be adopted. The UML was established as a standard "to model systems (and not just software)," and thus applies to views about software architecture.

The Unified Modeling Language (UML) allows analysts, software architects, and developers to specify, visualize, and document an entire system. By providing a common "language" for speaking about a complex system, the UML allows many different contributors, with diverse perspectives, to communicate on common ground. Key Rational scientists, including Grady Booch, Ivar Jacobson, and Jim Rumbaugh led the authoring of the UML standard. The Object Management Group (OMG) accepted Rational's submission of the UML and adopted it as the standard.

Grady Booch, Chief Scientist of Rational Software and one of the original developers of the UML, believes that UML is indeed an ADL because "…The UML supports multiple views of a system - both structural and behavioral [...]. Furthermore, there are many existence proofs for the use of the UML as an ADL: systems such as the AWACS mid-term modernization project and frameworks such as IBM's Insurance Application Architecture have used the UML to codify their architecture. That being said, there is value in considering the merging of the UML with traditional, more formal ADLs. Such a merger might bear fruit with regard to the formal analysis of a system's architecture and the direct execution of models"[1].

3.1. Architectural Representation

The architecture of the reference application is represented following the recommendations of the Rational Unified Process (RUP). The UML specification of the system has been divided into the following diagrams.

Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include:

Use Case Diagram displays the relationship among actors and use cases.

`Class Diagram` models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others.

Interaction Diagrams

- **Sequence Diagram** displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).
- **Collaboration Diagram** displays an interaction organized around the objects and their links to one another. Numbers are used to show the sequence of messages.

State Diagram displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli, together with its responses and actions.

`Activity Diagram` displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states. This diagram focuses on flows driven by internal processing.

`Physical Diagrams`

- **Component Diagram** displays the high level packaged structure of the code itself. Dependencies among components are shown, including source code components, binary code components, and executable components. Some components exist at compile time, at link time, at run times well as at more than one time.

- **Deployment Diagram** displays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units.

3.2. Extending UML for detailed description

UML can be used to model a business, prior to automating it with computers. The same basic UML syntax is used; however, a number of new symbols are added, in order to make the diagrams more relevant to the business process world. UML is designed to be extended in this way:

Stereotypes: Extensions to the syntax are created by adding 'stereotypes' to a model element. The stereotype creates a new model element from an existing one with an extended, user-defined, meaning. User defined symbols, which replace the original UML symbol for the model element, can then be assigned to the stereotypeStereotypes allow the UML syntax to be used to model anything.

Ways of representing a stereotype:

Place the name of the stereotype above the name of an existing UML element (if any).The name of the stereotype needs to be between «» (e.g. «node»)

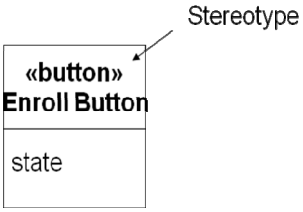


Fig 4: stereotype

Tagged values: Define additional properties for any kind of model elements. These can be defined for existing model elements and for stereotypes. These are shown as a tag-value pair where the 'tag' represents the property and the 'value' represents the value of the property. Tagged values can be useful for adding properties about

- code generation
- version control
- configuration management
- authorship

A tagged value is shown as a string that is enclosed by brackets {} and which consists of the tag, a separator (the symbol =), and a value (Figure 5).

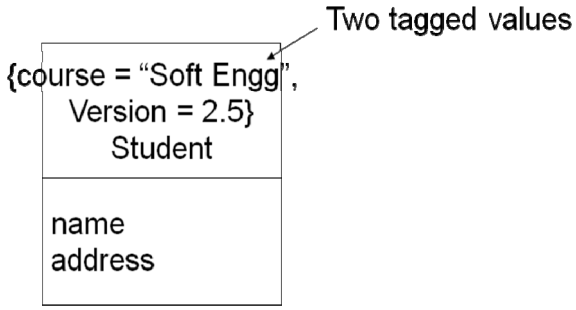


Figure 5: Tagged values

Constraints and Notes: Constraints are used to extend the semantics of UML by adding new rules, or modifying existing ones. Constraints can also be used to specify conditions that must be held true at all times for the elements of a model. **Comments** are used to help clarify the models that are being created e.g. comments may be used for explaining the rationale behind some design decisions. A comment is shown as a text string within a note icon.

Constraints and Notes: Constraints are used to extend the semantics of UML by adding new rules, or modifying existing ones. Constraints can also be used to specify conditions that must be held true at all times

for the elements of a model. **Comments** are used to help clarify the models that are being created e.g. comments may be used for explaining the rationale behind some design decisions. A comment is shown as a text string within a note icon (Figure 6).

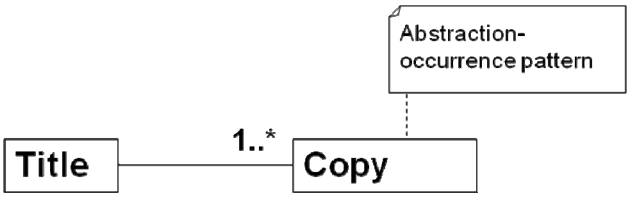


Fig 6: Constraints and Notes

4. Software Architecture Developed Using UML Provides The Following Benefits

- **Preliminary Analysis Model**
  Beyond the use case diagram, UML modeling can greatly help in the formalization of the preliminary analysis model. The modeling of business processes can be based on activity diagrams and the first models of your application can be realized using class diagrams. Objecteering/UML Requirements provides assistants to help in the creation of models from the requirements analysis and the dictionary. By using flow diagrams, Objecteering/UML Requirements provides you with a unique representation of the definition of your requirements through information flows exchanged between system elements
- **Component-Based Development**
  For the development of complex business systems, component-based development is the proven way to go. With Rational Rose, users can now model their components and interfaces even more effectively. Required components of a design can be easily reverse-engineered to explore the interfaces and interrelationships between other components in the model, lust drag and drop components from a file system onto the component diagram. Then you can reverse engineer, reuse, visualize, adapt, acquire, and create components for your system quickly and efficiently
- **Multi-Language and Multi-Platform Development**
  Large-scale software projects often involve a variety of languages. Rational Rose provides multi-language support so you can build components in mixed languages. Rose supports C++, Visual C++, lava, Visual Basic, Ada, and can generate Interface Definition Language (IDL) for CORBA applications, as well as Data Description Language (DDL) for database applications. Rational Rose is also platform independent, which means your team can work in Windows and UNIX environments
- **Complete Team Support**
  Rational Rose provides an easy way to coordinate development teams of ten to one hundred or more. Your analysts, architects, and engineers can operate in a private workspace that contains an individual view of the entire model, and modify his or her own piece of the model until it is ready to be merged with other models using the Model Integrator. Changes are made available to others by checking them into a configuration management and version control (CMVC) system. Rational Rose integrates with major CMVC tools including Rational ClearCase and Microsoft SourceSafe. Rose is also Tightly integrated with the Microsoft Repository and Unisys UREP and is open to other repositories.
- **Ease of Use**
  Rational Rose is the visual modeling tool of choice for many different types of software projects. In addition to using the UML, Rose supports familiar GUI elements like drag and drop with OLE automation. This lets you integrate with applications like Microsoft Word, and drive Rose from its OLE automation server interface. Rose's proven compatibility with the Windows platform has been tested in the Microsoft usability lab.
- *"A key benefit of UML is that users gain control over the architecture they are building. The key discriminator between successful and unsuccessful projects is how well architected the system is."*

5. UML AND OCL (Object Constraint Language)

The Object Constraint Language (OCL) is a notational language for analysis and design of software systems, which is used in conjunction with the Unified Modeling Language (UML) to specify the semantics of the building blocks precisely. OCL has been developed as a language for business modelling within IBM. It is part of the UML 2.0.

OCL can be used for a number of different purposes:

- To specify ***invariants*** on ***attributes*** and ***associations*** of classes and interfaces in a class model.
- To specify ***pre- and post - condition*** *on* ***operations*** in a class model
- To describe guard conditions on a state transition diagram.
- As query language

Compatibility of OCL and UML:

- Each OCL expression is written in the context of a UML model, a number of classifiers (classes), their features and associations, and their generalizations. All classifiers from the UML model are types in the OCL expressions that are attached to the model
- Within UML, types are organized in packages. OCL provides a way of explicitly referring to types in other packages by using a package-pathname prefix. The syntax is a package name, followed by a double colon:
  Packagename:: Typename
  This usage of pathnames is transitive and can also be used for packages within packages:
  Packagename1::Packagename2::Typename
- Any class in the UML model is a valid type in OCL
- All attributes of the class are also attributes of the class within OCL
- Constraints can be applied on the same association statement created in UML as shown in figure 8. OCL, a formal specification language that is part of the UML specification, enables you to annotate models with expressions that clarify meaning. In UML 1.1, the main purpose of OCL was to identify the constraints on model elements. For example, it could help you indicate that, to be assigned a room, a specific course must have at least six students enrolled. With OCL you could annotate the association between the Course and Classroom classes to represent the constraint, as shown in Figure 8. As an alternative to the note shown in this figure, you could use a constraint connector between the Course and Classroom classes [7].

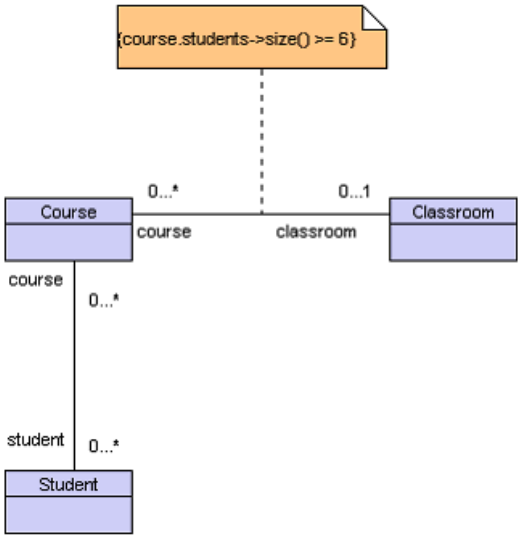


Fig 7: Using OCL to indicate a constraint on assigning a classroom to a course

6. Weaknesses of UML

There are two other shortcomings of the UML, but these can be addressed, either through the use of stereotypes or by imposing discipline on the way the UML is used.

- In the first case, the UML could be significantly improved by increased discipline in the use of relationship names. Most commonly a relationship name in the UML is a single verb that describes it

in one direction. Was this the only option, it would be unacceptable. It is, however, possible to add "roles" to each end of the relationship. This provides the ability to portray how an entity is viewed from the perspective of another entity. While no one outside the Barker world does this, it would be valuable if these role names were constrained to follow the Barker naming convention.

- Second, the UML only partially deals with unique identifiers. The philosophy behind object-orientation is that it isn't necessary explicitly to show unique identifiers. But then it turns out that from the point of view of a parent entity, it is often necessary to identify occurrences of a child entity. So "qualified associations" allow this to be expressed. But you are only allowed to identify an occurrence to a parent entity. You are not allowed to identify it to the world at large. This means that instead of a simple symbol attached to a relationship or attribute to indicate a unique identifier universally, you have to add a whole new box whose meaning is constrained and confusing at best.
- Comparing diagram files with each other: The current solution of providing a content merge viewer does not work for the "Compare With Each Other" use case when the selected files in the view part contain non-diagram ASCII files along with the diagram files (*.dnx). In this case, the compare/merge infrastructure within Eclipse determines the content merge viewer by inspecting the content of the files. Inspection of the content of the diagram file reveals that the file type is ASCII and therefore Eclipse creates a default text merge viewer. Though the default viewer allows you to edit the files and merge the individual differences, doing so could potentially corrupt the diagram file. Therefore, you should not edit any diagram files in the default viewer. Also, if you save a "Compare With Each Other" session with the diagram open, you get an error and an exception in the log file.
- UML is just syntax. It says nothing about how too create a model

7. Conclusion

Adapting UML to model software architecture is a practical step toward model driven development in which a system is developed via a sequence of model transformations that ends at an executable code. Using UML has the benefits of leveraging mainstream tools, skills, and processes. UML represents maturation in the development of object-oriented design notations. It offers a diverse collection of notations for capturing many aspects of the software development lifecycle, including not only traditional design concerns (such as functional decomposition), but also aspects of requirement analysis (particularly domain modeling), implementation, and testing (particularly scenario based functional testing). Our paper has described the benefits of UML standard and its compatibility and up gradation with other developing standards,on the basis of which UML can be adopted as a standard for developing software architecture. UML provides several extension mechanisms that allow you to do this without having to modify the underlying modeling language. These mechanisms let you add new building blocks, modify the properties of existing ones and even change their semantics.. Also some of the drawbacks of UML discussed. Current semantics of UML is quite loose. It needs to be made precise in order to provide more useful semantic information of applications. From the perspective of benefits and the extension mechanism, UML can be adapted as a standard language for software development architecture.

8. References

[1] Grady Booch, James Rumbaugh, Ivar Jacobson "The Unified Modeling Language User Guide"

[2] Frank Armour, Granville Miller "Advanced Use Case Modeling, Volume 2: Business Processes", Kogud School of Business at AmericanUniversity

[3] Object Management Group, OMG Unified Modeling Language Specification, Version 1.5

[4] Rational Software's Rational Home Page, http://www.rational.com

[5] Sinan Si Alhir, "UML in a Nutshell"

[6] Lochovsky, F., Lecture Notes on Software Engineering, Department of Computer Science, University of Washington, 1998, http://www.cs.washington.edu/education/courses/403/

[7] Ocl tools website: http://www.um.es/giisw/ocltools/