Issues with Representing Software Architectures in the Unified Modeling Language

SASA BASKARADA
School of Computer and Information Science
University of South Australia
Mawson Lakes SA 5095
AUSTRALIA
Sasa.Baskarada@unisa.edu.au

Abstract: - This paper shows how the Unified Modeling Language (UML) can be used to model software architectures effectively. Software architectures represent high-level views of systems and therefore allow developers to concentrate on the big picture rather than on low-level details [13]. They are also one of the best approaches to consider non-functional requirements early in the development process [1]. However, there is no standard definition of software architecture [17]. There is no agreement yet on how they should be modeled either. Therefore, software architectures are often described informally (using arrows, lines and boxes). Standardizing the notation, by using UML as a modeling tool, would reduce the ambiguity and make software architecture modeling easier for practitioners. A number of Architecture Description Languages (ADLs) already exist, which can be used to model software architectures. However, the software architecture community does not yet agree on what features should be present in an ADL, or precisely what these features should model [6]. Furthermore, there exists no common definition of the term Architecture Description Language either [17]. A large number of ADLs have been proposed as well and each of them takes a particular approach to the modeling of architectures [14]. This is one more reason why a standard modeling notation is needed. UML is already widely used for software analysis & design and it could also be a very useful tool for software architecture modeling. By modeling the CommPOS software architecture in UML, the paper shows how UML can be used to model component-based software architectures. However, further research into using UML for modeling of software architecture behaviors and dynamic software architectures is needed.

Key-Words: - Software Architecture, Software Design, Software Engineering, Unified Modeling Language (UML), Architecture Description Language (ADL)

1 Introduction

The Unified Modeling Language (UML) is an object oriented analysis and design language. It is a family of notations that has become a standard for developing software artifacts. It has also found application in modeling non-software artifacts, such as business processes, human workflows, and non-code development artifacts [9]. It is widely used in the software engineering community and there are many Computer-Aided Software Engineering (CASE) tools available that support UML modeling. Rational Rose, a popular graphical software modeling tool, uses the Unified Modeling Language (UML) as its primary notation [7].

Every system has an architecture and Clements and Northrop state that the term "software architecture" is mostly used to describe structural aspects of a software system [5]. They describe software architecture as an extremely important part of system design because it represents the earliest set of design decisions, which makes software architectures difficult to get right and hard to change [5]. Software architectures represent high-level views of systems and therefore allow developers to concentrate on the big picture rather than on low-level details [13]. They are also one of the best approaches to consider non-functional requirements early in the development process [1]. As the size and complexity of software systems grow, modeling software architectures is becoming ever more important. Unfortunately, there is no agreement yet on what software architectures exactly are or how to model them. Therefore, software architectures are often described informally (using arrows, lines, boxes, etc.), which can make them open to interpretation and hard to understand.

In an effort to standardize modeling of software architectures, a number of Architecture Description Languages (ADLs) have been developed (ACME, C2, Rapide, Wright, etc.). These languages were developed by a wide range of researchers and therefore there is no standard modeling language [6]. Furthermore, each ADL takes its own approach to modeling of software architectures. There is also no agreement yet on how exactly ADLs should model software architectures but there is a consensus that they should be able to at least model software architecture components, connectors and configurations. ADLs endeavor to make models of software architectures more understandable and enable a greater degree of analysis although some ADLs are more generic then others (some are specialized to particular domains).

UML has also been used to model software architectures, but it was originally designed to model object oriented analysis/design and therefore it may not be suitable for modeling all aspects of software architectures [11]. If UML was adapted to model software architectures, developers could use UML as a standard language for modeling software architectures as well as software analysis/design. Using UML would help visualizing, documenting and modeling of software architectures. This paper shows how UML could be used to model software architectures effectively.

2 Unified Modeling Language (UML)

UML is a language used for modeling object oriented analysis and design. It is a very large language which is defined by the Object Management Group (www.omg.org). The current UML specification document (version 1.5) has more than 700 pages. UML is a family of notations that includes use cases, use case diagrams, class diagrams, object diagrams, interaction diagrams, package diagrams, activity diagrams, statechart diagrams, component diagrams, and deployment diagrams. There are many Computer-Aided Software Engineering (CASE) tools available, which make it easy to draw UML diagrams (ArgoUML, Enterprise Architect, MagicDraw UML, Rational Rose, etc.). UML has become a standard language used for object oriented analysis and design. It is widely accepted in the software engineering community and having it as a standard language for the modeling of software architectures would make software architecture modeling easier for practitioners. UML notation is also a lot easier to understand than ADL notations.

3 Software Architectures

Software architectures are a key area in the software engineering discipline, because every system has an architecture. However, Rumpe et al state that there exists no common definition of the term software architecture [17]. There is no agreement yet on how exactly to model them either. The architecture depends on the requirements and the design decisions made to satisfy those requirements. A software engineer has to decide on the architecture of a software system in a similar way as a building architect decides on a particular architecture when building a house. The architecture can be newly developed or reused from similar existing systems. Certain types of applications have unique characteristics and share similar structure. Therefore, they might conform to the same architecture. In order to speed up development, reduce production cost, control the complexity, elevate abstraction levels, achieve separation of concerns and facilitate software reuse, certain software architectures have been developed for certain types of applications. Therefore, developing applications affirming to particular software architectures potentially increases productivity and reliability. Software architectures provide a template for design and also allow the management to better estimate the costs involved in the project. Unfortunately, software architectures are often described informally (using arrows, lines, boxes, etc.), which can make them open to interpretation and hard to understand. Software architectures aid in building a system by structuring large collections of components (clients, servers, databases, etc). They focus on the system structure and interaction between different components. This is one of the most important aspects of large system design. Bachman et al state that, one cannot hope to build an acceptable system unless the architecture is appropriate and effectively communicated [3]. There is still little consensus on software architecture terminology, representation and methodology [5], which makes the modeling more difficult. The term itself “Software Architecture” seems to be overused at the moment (it is a buzz word). Therefore, it is used to describe various things, many of which are not software architectures at all. Clements & Northrop also describe software architecture as a vehicle for stakeholder communication, because it provides a common ground for discussing concerns among different stakeholders [5]. Garlan states that software architecture plays an important role in the understanding, reuse, construction, evolution, analysis and management of software systems [8].

There are two basic approaches to architecting software systems. The top-down approach divides a large problem into a number of sub-problems, which can then be newly implemented or solved by reusing existing components. The second approach is bottom-up. This approach requires implementing new components or reusing existing ones to compose a system. Real life situations require the use of both of these approaches. This is important because software architectures can be seen as being composed of components, connectors and configuration. We can get those by decomposing the system (or we can use the bottom-up approach to compose a system out of existing components, connectors and configurations). This paper shall therefore also investigate if any UML diagrams (class diagrams, collaboration diagrams, etc.) could be used to represent software architectures by modeling components, connectors and configurations.

3.2 Architectural Views

Clements et al state that modern software architecture practice embraces the concept of architectural views, and Bachmann et al state that documenting software architecture is primarily about documenting the relevant views [4], [3]. Views are essentially abstractions, each with respect to different criteria [5]. Many different views can be used to model software architectures, but Kruchten describes the "four plus one" approach [10]. He identifies four main views of software architecture plus a fifth view that ties the other four together. Bachmann et al describe these views as the logical view (behavioral requirements, and the services the system should provide to its end users), process view (performance, system availability, concurrency, distribution, system integrity and fault tolerance), development view (the actual software models), and physical view (system availability, reliability, performance and scalability) [3].

3.2.1 The Layered View

One of the most commonly used views in software architectures is the layered view [3]. Bachmann et al also describe a layer as a collection of software units such as programs or modules that may be invoked or accessed [3]. It is mostly represented as vertically arranged rectangles. Layering divides the software structure into discrete units (presentation layer, application layer, data layer, etc.). Each layer provides functionality, which is independent from any other layers. Therefore, a layer can be considered a component in software architecture modeling. Each layer also has an interface, which can be used by other layers. Bachmann et al defines an interface as a boundary across which two independent entities meet, interact, or communicate with each other [2]. Therefore, as long as layer's interface is not changed, a layer can be modified without affecting any other layers. Bachmann et al also state that UML has no built in primitive corresponding to a software architecture layer, but layers could be defined as a stereotype of a UML package [3].

3.3 Components

Software architectures are also often described as models of components and interconnections among these components. Medvidovic & Taylor describe a component in an architecture as a unit of computation or a data store (a familiar example is a Unix process) [12]. They can vary greatly in size and even a software architecture layer can be considered as a component. Egyed & Kruchten suggest the use of UML class diagrams for modeling of component based architecture [7]. However, other UML diagrams could be used as well. Components maintain state, perform operations and exchange messages with other components [11].

3.4 Connectors

Connectors model component interactions and the rules governing those interactions. They transmit messages between components. Simple interactions can be achieved through method calls and global variables. More complex interactions include database access and client-server applications. Egyed & Kruchten suggest the use of UML aggregations, associations, dependencies and generalization as connectors [7]. These UML connectors could have stereotypes or constraints associated with them as well.

3.5 Configurations

Configurations are instances of components and connectors. Medvidovic & Taylor describe configurations as connected graphs of components and connectors that describe architectural structure [12]. They describe semantics of a software architecture and place constraints on component interaction.

4 Architecture Description Languages (ADLs)

ADLs are not programming languages, but rather languages used to model software architectures. Rumpe et al state that there exists no common definition of the term architecture description language (ADL) and that there is no standard ADL [17]. Some of the ADLs include ACME, Aesop, ArTek, C2, Darwin, LILEANNA, MetaH, Rapide, SADL, UniCon, Weaves, Wright, etc. ADLs attempt to formalize modeling of software architectures and they are formal notations for modeling the structure and behavior. Tools are also available, for many ADLs, which support visual representation and analysis.

There is a large number of ADLs and there is no standard notation yet. Furthermore, each ADL takes a particular approach to modeling (each ADL addresses a particular problem domain), which means that most ADLs can only be used to model a particular set of architectures. Dashofy et al state that research and experimentation in software architectures have resulted in an overabundance of ADLs [6]. These ADLs are mostly developed and used in academic circles and they haven't yet gained much acceptance in practitioners' community. ADLs are not used to a large extent by the developers mostly because there is a no standard ADL and the syntax is also fairly complex. There is also no agreement on which exact features an ADL must support, but there is an acceptance that they have to provide notations for modeling software architecture components, connectors and configurations.

5 UML as an ADL

I have highlighted the problem that there is still extensive disagreement about what software architecture really is and how to model it. There is also no agreement yet on what precisely an ADL is either. Each ADL takes a particular approach to modeling of software architectures and there is a broad variety of ADLs (there is no standard language). One of the solutions to these problems is to use UML as an ADL. If UML is to be used as an ADL, UML would have to take its own approach to the modeling of software architectures. It would not make sense to try to imitate all the features of existing ADLs, but to select the features which are considered as absolutely necessary in an ADL. There seems to be consensus in the research community that an ADL has to be able to at least model components, connectors and configurations. Therefore, if UML is to be used as an ADL, it would have to be able to model these successfully.

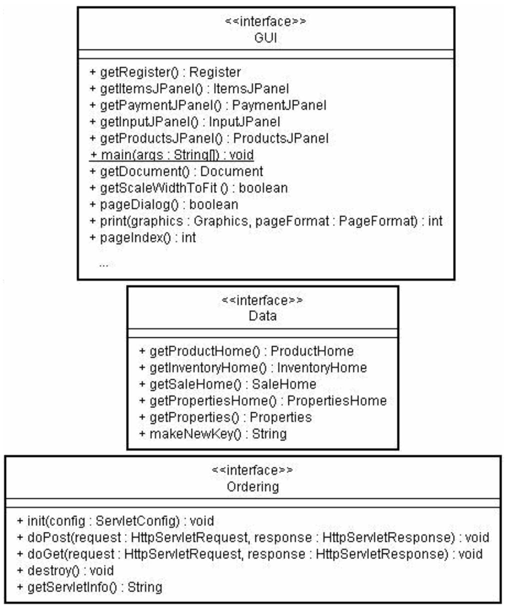
Various UML diagrams could be used to model software architecture components; however UML explicitly provides Component diagrams for component modeling. UML Component diagrams describe the organization of components in the system. Use cases can be used to describe the components in more detail and to specify the component functionality. Various other diagrams (class diagrams, interaction diagrams, component diagrams, etc.) could also be used to represent components visually. UML state diagrams can be used to represent component states and package diagrams can be used to represent groupings of components (packages can also be used to model architectural layers by grouping various components). UML interfaces, which are collections of operations, can be used to specifically model component interfaces. UML realizations, associations or dependencies can be used to model connectors. The choice would depend on the type of the connector and on the diagram used to represent software architecture components. As this paper is concentrating on the use of component diagrams for representation of components, dependencies are going to be used to model software architecture connectors. UML constraints could be used to specify constraints on component interaction. These constraints would preferably be specified in the Object Constraint Language (OCL). Pre and post conditions could be specified in OCL as well.

5.1 Extensions to UML based on ADLs

Medvidovic et al suggest some lightweight extensions to UML based on C2, Rapide and Wright [11]. The UML extensions suggested by the authors attempt to provide UML with all the features that these ADLs support. Extending UML based on their work essentially creates a new ADL, which contains all the features of C2, Rapide and Wright combined. Having all these extensions may not be necessary, because each one of these ADLs takes different approach to the modeling of software architectures. C2, Rapide and Wright are essentially not compatible with each other in the first place and if UML is to be used as an ADL, UML would not have to be compatible with all of them either. Pérez-Martínez states that the only solution to modeling the C3 architectural style (C3 is derived from C2) in UML is to extend the UML meta-model [16]. In reality extending the UML meta-model would not be practical because the resulting language would not be compatible with existing CASE tools. Therefore, this paper suggests lightweight extensions (stereotypes, constraints and tagged values) to UML for representation of software architectures.

6 CommPOS Software Architecture in UML

In order to investigate UML's ability to model software architectures, I have used UML to model the software architecture of the Commercial Point Of Sale System (CommPOS v2.0). CommPOS is a point of sale system developed by myself, Damien Presser and Matthew Bauerochse. It is a large system with a non trivial software architecture and I have successfully modeled it in UML (by modeling components, connectors and configurations). UML Component elements have been used to represent various system components and UML Interfaces have been used to specify the interfaces of these components. Stereotypes were used to further specialize the meaning of individual components and OCL was used to specify any constraints on component communication. UML dependencies were used to model connectors. Since CommPOS already existed (the system was already developed), in order to model its architecture, the system needed to be decomposed into components. These components were then used to model the software architecture of CommPOS. Since components can vary in size (depending on the granularity level), there was not only one correct way to model CommPOS architecture. However, by examining the system I could, among others, identify following components: JBoss (Java application server), Data (Enterprise Java Beans), Ordering (Java Servlet used for web access), GUI (the presentation layer), and Store (the physical store). Component interfaces were specified in a UML class diagram (Fig.1).



<<interface>>
GUI
+ getRegister() : Register
+ getItemsJPanel() : ItemsJPanel
+ getPaymentJPanel() : PaymentJPanel
+ getInputJPanel() : InputJPanel
+ getProductsJPanel() : ProductsJPanel
+ main(args : String[]) : void
+ getDocument() : Document
+ getScaleWidthToFit () : boolean
+ pageDialog() : boolean
+ print(graphics : Graphics, pageFormat : PageFormat) : int
+ pageIndex() : int
...
<<interface>>
Data
+ getProductHome() : ProductHome
+ getInventoryHome() : InventoryHome
+ getSaleHome() : SaleHome
+ getPropertiesHome() : PropertiesHome
+ getProperties() : Properties
+ makeNewKey() : String
<<interface>>
Ordering
+ init(config : ServletConfig) : void
+ doPost(request : HttpServletRequest, response : HttpServletResponse) : void
+ doGet(request : HttpServletRequest, response : HttpServletResponse) : void
+ destroy() : void
+ getServletInfo() : String

Fig.1 Component Interfaces (partial diagram).

Next, I modeled the components and connectors using a UML Component diagram (Fig.2). Dependencies were used to represent Connectors.

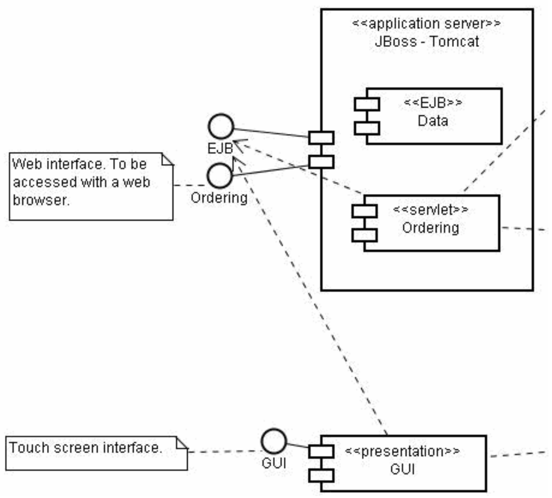


Fig.2 Components and Connectors (partial diagram).

Finally, I used OCL to specify constraints as well as pre conditions and post conditions.

7 Conclusion

There is no consensus in the research community on what software architectures really are and how to represent them. Software architectures are often represented informally by using boxes, lines and arrows. They are also often represented by using views (an example is the layered view). ADLs take the approach to the representation of software architectures by modeling components, connectors and configurations (although many ADLs provide many other features as well).

Many researchers try to tackle these issues by making their own definitions and creating their own ADLs for representation of software architectures. That has resulted in a large number of different ADLs, each taking a different approach to the modeling of software architectures and each having a different syntax and semantics. There is no standard ADL yet and therefore ADLs haven't gained much acceptance in the developers' community. Medvidovic et al and Pérez-Martínez have done research in using UML to model software architectures, but their research concentrates on using UML to mimic the features of existing ADLs [11], [16]. Existing ADLs are not compatible with each other in the first place and if UML is going to be used as an ADL, it wouldn't have to be compatible with other ADLs either. This paper has presented an approach to modeling software architectures in UML by modeling components, connectors and configurations.

8 Future Work

This paper did not specifically address behavioral aspects of software architectures found in many ADLs. Even though UML state diagrams could be used to represent states of individual components, they may not suffice for modeling of inter-component behaviors. Various other diagrams, object diagrams, activity diagrams, sequence diagrams and communication diagrams could possibly also be used to model architecture behavior. Another area requiring further research is the representation of dynamic software architectures in UML. Some ADLs (Rapide for example) allow for the modeling of software architectures in which the number of components, connectors and configurations may vary over time.

References:

[1] Andersson, J, 'Issues in Dynamic Software Architectures', in *Proceedings of the 4th International Software Architecture Workshop ISAW-4*, Limerick, Ireland, 2000, pp. 111-114.

[2] Bachmann, F, Bass, L, Clements, P, Garlan, D, Ivers, J, Little, R, Nord, R & Stafford, J, 'Documenting Software Architecture: Documenting Interfaces', CMU/SEI-2002-TN-015, Software Engineering Institute, Carnegie Mellon University, 2002.

[3] Bachmann, F, Bass, L, Carriere, J, Clements, P, Garlan, D, Ivers, J, Nord, R & Little R, 'Software Architecture Documentation in Practice: Documenting Architectural Layers', CMU/SEI-2000-SR-004, Software Engineering Institute, Carnegie Mellon University, 2000.

[4] Clements, P, Garlan, D, Little, R, Nord, R & Stafford J, 'Documenting software architectures: views and beyond', in *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, United States, 2003, pp. 740-741.

[5] Clements, PC & Northrop, LM, 'Software Architecture: An Executive Overview', CMU/SEI-96-TR-003, ESC-TR-96-003, Software Engineering Institute, Carnegie Mellon University, 1996.

[6] Dashofy, EM, Hoek, A & Taylor, RN, 'An infrastructure for the rapid development of XML-based architecture description languages', *in Proceedings of the 24th International Conference on Software Engineering*, Orlando, Florida, USA, 2002, pp. 266-276.

[7] Egyed, A & Kruchten, PB, 'Rose/Architect: A Tool to Visualize Architecture', in *Proceedings of the 32nd Annual Hawaii Conference on Systems Sciences*, vol. 8, 1999, p. 8066.

[8] Garlan, D, 'Software architecture: a roadmap', *in Proceedings of the 22nd international Conference on Software Engineering, The Future of Software Engineering ICSE2000*, Limerick, Ireland, 2000, pp. 91-101.

[9] Kruchten, P, Kozaczynski, W & Selic, B, 'ICSE 2001 workshop on describing software architecture with UML', *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 6, 2001, pp. 78-79.

[10] Kruchten, P, 'The 4 + 1 View Model of Architecture', *IEEE Software*, vol. 12, no. 6, 1995, pp. 42-52.

[11] Medvidovic, N, Rosenblum, DS, Redmiles, DF & Robbins, JE, 'Modeling software architectures in the Unified Modeling Language', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 1, 2002, pp. 2-57.

[12] Medvidovic, N & Taylor, RN, 'A Classification and Comparison Framework for Software Architecture Description Languages', *IEEE Transactions on Software Engineering*, vol. 26, no. 1, 2000, pp. 70-93.

[13] Medvidovic, N, Rosenblum, DS & Taylor, RN, 'A Language and Environment for Architecture-Based Software Development and Evolution', in *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, CA, USA, 1999, pp. 44-53.

[14] Medvidovic, N & Rosenblum, D, 'Domains of Concern in Software Architectures and Architecture Description Languages', in *Proceedings of the 1997 USENIX Conference on Domain-Specific Languages*, Santa Barbara, California, 1997.

[15] Object Management Group, viewed 20 September 2004, <http://www.omg.org>.

[16] Pérez-Martínez, JE, 'Heavyweight extensions to the UML metamodel to describe the C3 architectural style', *ACM SIGSOFT Software Engineering Notes*, vol. 23, no. 3, 2003, p. 5.

[17] Rumpe, B, Schoenmakers, M, Radermacher, A & Schurr, A, 'UML+ROOM as a standard ADL?', in *Proceedings of the 5th IEEE International Conference on Engineering of Complex Computer Systems ICECCS '99*, Las Vegas, Nevada, United States, 1999, pp. 43-53.