Documenting Software Architectures using UML

FILIP PRENTOVIĆ, GTECH, Belgrade
ZORAN BUDIMAC, University of Novi Sad

As software systems become large and more complex, focus on main design issues is shifted from algorithms and data structures. Software architecture, which represents high-level organization of software system, brings whole new set of design issues: overall system organization, global control structures, communication protocols, data access and synchronization, as well as choosing between different design solutions. In this paper, ways of using UML to document component and connector views are described. Following elements of component and connector view will be described using UML: components and component types, connectors and connector types, ports, roles, systems and properties.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures

General Terms: software architecture, UML

Additional Key Words and Phrases: architecture description languages, component and connector view

1. INTRODUCTION

As software systems become large and more complex, focus on main design issues is shifted from algorithms and data structures. Software architecture, which represents high-level organization of software system, brings whole new set of design issues: overall system organization, global control structures, communication protocols, data access and synchronization, as well as choosing between different design solutions.

Although architectural descriptions of software systems play important role in software design, they are often represented in informal way, by using simple diagrams with ad-hoc notation, with little or no influence on later phases of software development. To overcome this, a number of architecture description languages (ADLs) have been developed, in order to improve understandability, reusability and analysis capabilities of architectural descriptions. Also, ADLs are formal way of representing architectures, they permit analysis and assessment of architectures, for completeness, consistency, ambiguity, and performance, and, in some cases, can support automatic generation of software systems [Clements et al., 2002].

Despite aforementioned advantages of ADLs, there are several disadvantages of using ADLs, which limited their use in practice. First and foremost, there is lack of support by commercial tools, primarily because there's no universal agreement on what ADLs should represent. Because of this, UML was proposed as a solution for describing software architectures. UML has many clear advantages: large number of commercial tools, widespread use, as well as implementation capabilities.

In this paper, possible strategies for representing component and connector view in UML are presented- Section 2. Section 2.1 contains possible ways of describing architectural components, section 2.2 presents strategies for description of ports, section 2.3 depicts some possible solutions for describing architectural connectors using UML concepts, while strategies for describing systems and properties are presented by sections 2.4 and 2.5, respectively. Related work is provided by section 3. Conclusion and future work are given by section 4. Glossary of terms used in this paper is given by section 5.

2. DOCUMENTING COMPONENT AND CONNECTOR VIEWS

In this section, ways of using UML (class diagram in particular) to document component and connector view are described. Following elements of component and connector view will be described using UML:

Author's address: F. Prentović, Bul. M. Pupina, 11000 Belgrade, Serbia; email: filipprentovic@yahoo.com; Z. Budimac, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia; email: zjb@dmi.uns.ac.rs.

In: Z. Budimac (ed.): Proceedings of the 2nd Workshop of Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA), Novi Sad, Serbia, 15.-17.9.2013, published at http://ceur-ws.org

components and component types, connectors and connector types, ports, roles, systems and properties [Garlan et al., 2002].

Since there is no best way of documenting these elements using UML, multiple alternatives will be presented, along with their advantages and disadvantages. Architectural descriptions which are produced this way should respect documented UML semantics and the intuitions of UML modelers. The interpretation of the encoding UML model should be close to the interpretation of the original architectural description so the model is intelligible to both designers and UML-based tools. Also, resulting architectural descriptions in UML should bring conceptual clarity to a system design, avoid visual clutter, and highlight key design details. All relevant architectural features for the design should be represented in the UML model, keeping in mind that not all uses of UML are supported equally by all UML tools, which can limit documentation options [Clements et al., 2002].

It's worth pointing out that strategies for documenting component and connector view in UML presented in this paper rely heavily on concepts introduced in UML 2.0, such as structured classifiers and composite structure diagrams. A composite structure diagram depicts the internal structure of structured classifiers (e.g. classes and components) by using parts, ports and connectors, and some of these concepts appear as a natural candidate for appropriate architectural elements. For the explaining purposes, simple publish-subscribe architecture, in which publisher and subscribers communicate through the event dispatcher (represented as event "bus"), will be considered (Figure 1) [Clements et al., 2002].

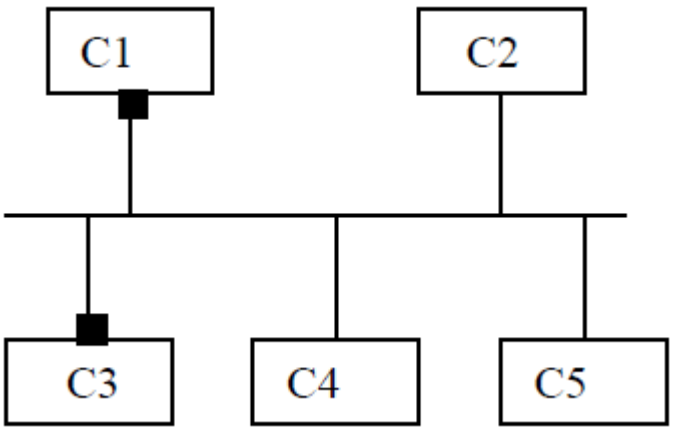

Figure 1. Simple publish-subscribe architecture

2.1 Documenting components

There are two meaningful ways of representing architectural components in UML, by using UML classes or UML components.

A natural candidate for representing component types in UML is the class concept. Classes describe the conceptual vocabulary of a system just as component types form the conceptual vocabulary of architectural documentation. Also, UML classes, like component types in architectural descriptions, are first-class entities and rich structures for capturing software abstractions.

Structural properties of architectural components can be represented as class attributes or associations, behavior can be described using UML behavioral descriptions, and generalization can be used to relate a set of component types. The type/instance relationship in architectural descriptions is a close match to the class/object relationship in a UML model, although it's not identical. A component instance might refine the number of ports specified by its type or associate an implementation in the form of an additional structure that is not part of its type's definition. This can be overcome by using subclasses, so that instance of a subclass represents instance of a component.

On the other hand, UML components have an expressive power similar to that of classes and can be used to represent architectural components as in the first strategy, with slightly different graphical notation. In addition, component can own packageable elements along with elements a class can own, which can be useful in some cases.

Choosing between these two strategies may rely more on the semantic match offered by them, since both have nearly the same expressiveness, and are visually nearly identical. One thing which can influence the decision is a way in which connectors are documented. For example, classes can be used to describe connectors, in which case using classes for describing components can have negative impact on visual clarity of architectural description.

2.2 Documenting ports

Most natural way of representing architectural ports in UML is using port concept, introduced along with classifier concept. UML ports are explicit interaction points for classifiers, so it can be used with both classes and components. UML ports provide all the expressiveness needed to document architectural ports. Multiple ports can be defined for a component type, enabling different interactions. Since ports in UML can be typed by an interface or a class, multiple instances of the same port type are allowed. Ports can also be associated with multiple interfaces, provided and required (Figure 2).

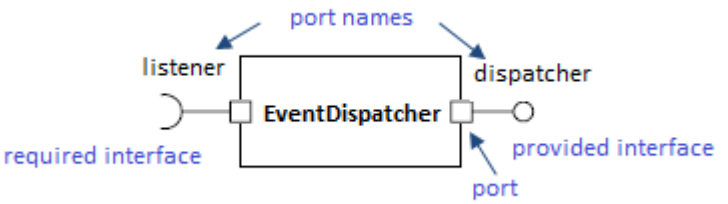

Figure 2. UML class with ports

Ports can be omitted in some diagrams, in case of single port components, or if the ports can be inferred from the system topology. Identifying the ports of a component allows different interactions to be distinguished based on the port through which they occur.

2.3 Documenting connectors

Even though there is connector concept in UML, it lacks expressiveness needed to be considered as a solution for documenting architectural connectors, e.g. it lacks ability to associate semantic information with a connector or to clearly document architectural connector roles. Therefore, three strategies for representing connectors in UML will be considered:

- ☐ using UML associations
- ☐ using UML association classes
- ☐ using UML classes

Using UML associations for documenting connectors allows visual distinction between components and connectors. Different types of connectors can be distinguished by labeling each association with a stereotype (Figure 3).

This strategy lacks ability to express semantic information connectors provide, because roles of connector can't be defined by associations (since associations don't have UML interfaces or ports), and associations can't own neither attributes nor behavioral descriptions. Nevertheless, this strategy is useful when purpose of documentation is identifying where different types of connectors are used in a system.

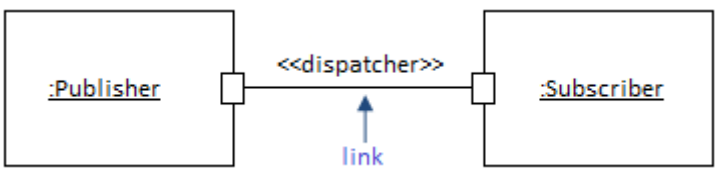

Figure 3. Documenting connector using link

The second strategy consists of using UML associations class for describing connectors (Figure 4). This strategy allows semantic descriptions of connectors, by using attributes and behavioral descriptions of a class, as well as creating substructure of connectors, if needed.

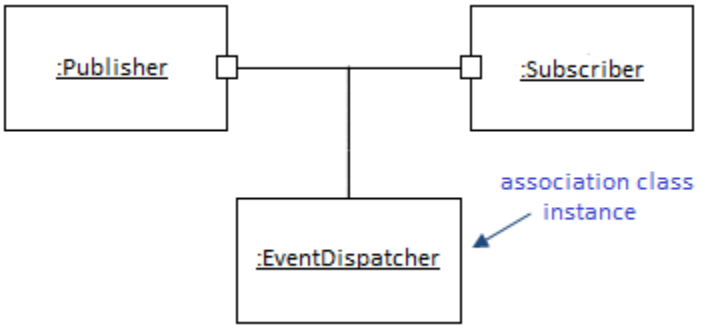

Figure 4. Documenting connector using association class instance

Also, connector roles now can be represented as UML ports on the association class, which enables same level of expressiveness that component ports have (Figure 5).

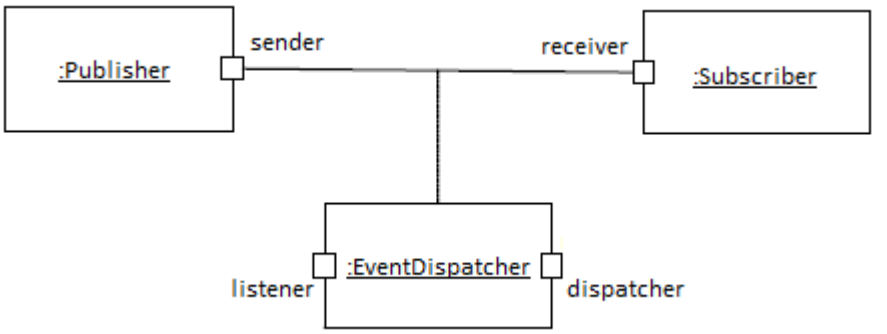

Figure 5. Documenting connector using association class instance with ports

However, this brings out issue of attaching component ports to connector roles. This can be solved either by role name being added to corresponding component port, or by using assembly connectors between ports of objects representing the connector and the ports of the objects representing the components. First solution doesn't affect visual clarity, but it lacks standard tool support, and second solution introduces substantial visual clutter.

The third strategy consists of using UML class for documenting connectors (Figure 6).

Figure 6. Documenting connector using object

This strategy allows same expression capabilities as the previous one, but also resolves the component and connector attachment problems by explicitly using UML assembly connectors, removing the potential ambiguity of the second strategy. Unfortunately, this solution presents the poorest visual distinction between components and connectors, especially if classes are used for documenting components. This problem can be mitigated by using different UML concepts to represent components and connectors, as shown in Figure 7, where UML components are used for representing architectural components, and class is used for representing connector.

Figure 7. Documenting connector using object, UML components document components

However, this variation is only slightly better in terms of visual distinctiveness. Another solution to this problem would be using UML stereotype mechanism, which allows customized visualization of stereotyped elements (Figure 8). However, this solution has limited practical application, since use of stereotypes requires graphical support not offered by most UML tools.

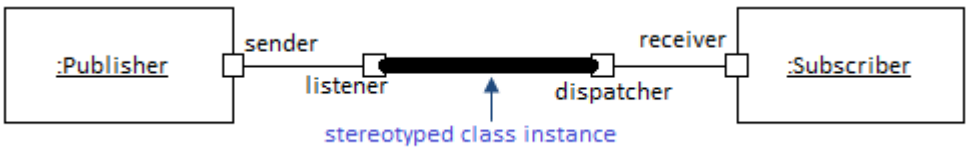

Figure 8. Documenting connector using stereotyped class

Decision on which strategy is the most suitable for documenting connectors should be made depending on answers for following questions:

- ☐ Does the architectural description need only types of connector being identified, and/or effects of connector on component interaction?
- ☐ For which phase of system's development lifecycle architectural description is being written?
- ☐ Which strategy is being used for documenting components?
- ☐ What tool support is available?

If the goal of documentation is to identify where different types of connectors are used in a system, particularly if the connector types are well known and understood, first strategy is a good choice. This strategy is a good choice for "first drafts", in which specific connector semantics have not been defined, but crude choices should be identified by name. Drawback of this strategy is its inability to describe semantics and role of a connector. Second strategy is a good choice when connector semantics need to be described, but specific component port and connector role attachments are not important. If these attachments are important, third strategy can be used.

2.4 Documenting systems

Documenting systems relies on definitions of component and connector types, as well as topologies of instances of component and connector types. Types and their hierarchies are documented using class and component diagrams. For subtypes generalizations can be used, and for types decomposition classes can be used. Topologies of component and connector instances can be documented using instance diagrams.

2.5 Documenting properties

Architectural properties capture semantic information about a system. Therefore, UML properties concept can be used for describing them, because UML properties represent structural feature. There are three possible ways of documenting properties in UML:

- ☐ using tagged values
- ☐ using attributes
- ☐ using stereotypes

Tagged value is name-value pair that can be used for documenting information semantically relevant to classifier, and therefore can be used to document architectural properties. Downsides of this approach are that there is no explicit documentation of the value's type, and that tagged value can be defined only for instance.

UML attributes can't be used directly for describing properties because they represent structural elements. This can be overcome by using stereotype denoting that an attribute is semantic, not structural.

Third strategy consists of using stereotypes as a way of extending UML meta model and allow new semantics to be incorporated into UML models. One way to extend a concept is to define a stereotype that includes a tag definition. When the stereotype is applied, the value of a tagged definition is called a tagged value.

Each of these strategies has advantages over the others. If analysis tools are not used and properties are not required for all instances of a class, the first strategy is adequate. The second strategy is a good choice if the properties are mandatory to support analysis, but the implementation consequences are not

terribly detrimental. The third strategy also provides explicit documentation of the property type, but lacks the semantic mismatch and potential implementation consequences of the second strategy.

3. RELATED WORK

Concepts introduced in UML 2 are used in [Anacleto 2008], where UML profile and a group of UML patterns for documenting the component and connector view of software architectures are presented. In this work, UML components are used for representing connectors, with stereotypes used for visual distinction from architectural components represented by UML components as well. A different approach is considered in [H. B. Christensen et al., 2011], with component instances represented as UML objects, and connector instances represented as links. Some aspects of hierarchical decomposition of a system into modules in UML 2 are discussed in [Frick et al., 2004].

Apart from strategies for documenting component and connector view described previously, attempts were made in order to describe different aspects of software architectures using concepts that existed in UML 1.x. Using UML 1.x in documenting different architectural views is considered in [Hofmeister et al., 1999]. Although the conclusion was that UML works well for describing a static structure of the architecture, it also emphasized its lack of support for describing ports on components, among other things. Constraining and extending UML 1.x meta model is considered in [Medvidovic et al., 2002], in addition to using existing UML notation. Constraining UML meta model seems appropriate in trying to enforce architectural constraints, while augmenting meta model could be beneficial in trying to incorporate new modeling capabilities. Component-modeling capabilities of UML are presented in [Kobryn 2000], with the emphasis on modeling component frameworks like EJB and COM+, which shows that UML 1.3 provides basic support for this type of modeling, with issues that comes from semantics overlap between components and classes, as well as lack of support for large component systems and frameworks.

4. CONCLUSION AND FUTURE WORK

In this paper, multiple strategies have been presented for documenting elements of architectural descriptions in UML, along with guidelines for determining which strategy represents the best solution. It is obvious that UML 2 is more suitable for describing component and connector view of software architecture than its predecessors. However, there are a few issues that continue to make documenting architectures relatively complex: UML connectors are not first class entities, so less natural representations of architectural connectors must be used, and there isn't a completely natural way of documenting architectural properties. Also, concept of a part, introduced with composite structure diagrams in UML 2 can be taken into consideration when showing composite properties of the containing structured classifier. Furthermore, documenting other views of software architectures in UML can be considered, along with more complex and more illustrative examples.

5. GLOSSARY

Since software architecture has many definitions, and since architectural descriptions differ depending on a phase of software development life-cycle in which they are used, a glossary is needed in order to avoid confusion, and also to explain terms with potentially ambiguous meaning. Here is the list with short definition of terms which are used throughout the paper:

- software architecture - structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them [Clements et al., 2002]
- architecture description languages (ADLs) - language and/or a conceptual model to describe and represent system architectures
- component and connector view - define models consisting of elements that have some run-time presence, such as processes, objects, clients, servers, and data stores. Additionally, component and

connector models include as elements the pathways of interaction, such as communication links and protocols, information flows, and access to shared storage [Clements et al., 2002].

- components and component types - represent the principle runtime elements of computation and data storage such as clients, servers, filters, and databases
- connectors and connector types - represent the runtime pathways of interaction between components such as pipes, publish-subscribe buses, and client-server channels
- component interfaces (or ports) - represent points of interaction between a component and its environment
- connector interfaces (or roles) - represent points of interaction between a connector and the components to which it is connected
- systems - graphs representing the components and connectors in the system and the pathways of interaction among them
- properties - additional information associated with structural elements of an architecture
- styles - define a vocabulary of component and connector types together with rules for how instances of those types can be combined to form an architecture in a given style e.g. pipe-and-filter, client-server, and publish-subscribe

REFERENCES

L. V. A. Anacleto, A UML Profile for Documenting the Component-and-Connector Views of Software Architectures. Epidata Consulting, April 2008.

P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond. Boston, MA: Addison-Wesley, 2002.

H. B. Christensen, A. Corry and K. M. Hansen, The 3+1 Approach to Software Architecture Description Using UML, Department of Computer Science, University of Aarhus, 2011

G. Frick, B. Scherrer and K. D. Muller-Glaser, Designing the Software Architecture of an Embedded System with UML 2.0. Software Architecture Description & UML Workshop, October 2004.

D. Garlan, S. Cheng and A. J. Kompanek, Reconciling the Needs of Architectural Description with Object Modeling Notations. Science of Computer Programming, Special UML Edition 44, 1 (July 2002): 23-49.

C. Hofmeister, R. L. Nord and D. Soni, Describing Software Architecture with UML. First Working IFIP Conference on Software Architecture, February 1999.

C. Kobryn, Modeling Components and Frameworks with UML. Communications of the ACM, vol. 43, no. 10, October 2000.

N. Medvidovic, D. Rosenblum, D. Redmiles and J.Robbins, Modeling Software Architecture in the Unified Modeling Language. ACM Transactions on Software Engineering and Methodology (TOSEM) 11, 1 (January 2002): 2-57.