KNOWLEDGECUDDLE PUBLICATION

International Journal of Computer Engineering and Science, August- 2014

Documenting Software Architectural Component and Connector with UML 2

Vikram M. Agrawal
IT Department
BVM Engineering College
agrawal.vm@gmail.com

ABSTRACT: ***Earlierversions of the UML have been an out of depth for documenting software architectures like component, port, connector and system. Users have adopted conventions for representing architectural concepts using different grouping of UML modeling element. They can also create profiles to focus the UML. Changes incorporated in UML 2 have improved UML's suitability for software architectural documentation, but UML is still an out of your depth for documenting some types of architectural information. In this paper, there is description of documenting component and connector using UML but in particular case, documenting architectural connectors and components remains problematic.***

Keywords: - **component, connector**

---

1. INTRODUCTION

The first work of stakeholder to create communicating architecture of whole system because architectures are rational constructs of enduring and long-lived importance.Architecture must be clearly explained because others are to build systems from it, analyze it, maintain it, and learn from it. Therefore, increased attention is now being paid to how architectures should be documented.

Modern software architecture practice embraces the concept of architectural views as part of the solution [1–4]. A view is a representation of a set of system elements and relations associated with them [5]. Systems are typically recognized in terms of many views, each of which represents aspects of the system needed to analyze and communicate different quality attributes of the system. So we can consider the all the view of all the stakeholders. For instance, a layered view is useful for understanding modifiability, while a communicating processes view is useful for understanding system performance.

The widespread presence of UML has directed practitioners to try to use it to document software architectures. The results have been mixed and somewhat unacceptable and un-useful. For example, UML has no built-in way to represent the basic architectural concept of a layer. Nor was there any straightforward way to represent a connector, in the rich sense proposed by Garlan and Shaw [6]. Nevertheless, ways have been proposed to represent several familiar architectural views using UML. Previous work has investigated ways that it can be used as is to represent architectural concepts or ways that UML can be specialized to improve its suitability for architectural documentation, such as [7–9].

During the development of UML 2, changes were made that significantly improve UML's suitability for architectural documentation [10]. Structured Classifiers permit improved representation of architectural hierarchy, often used effectively to represent detail in stages or for different USERS. Ports provide a normal way to represent runtime points of interaction and collections of interfaces involved in a common protocol.

Unfortunately, some of UML's drawback with respect to architectural documentation remain. In this paper, we examine one such drawback of documenting architectural connector with UML 2. Though UML 2 provides better support for representing connectors, users are still left with a variety of modeling options to choose among, each of which is best suited to different uses.

To distinguish between architectural and UML uses of the same term, we capitalize the names of UML modeling elements (e.g., Component or Class) and use lower case for architectural concepts (e.g., component or connector).

2. COMPONENT AND CONNECTOR VIEWS

Component and connector views present architecture in terms of elements that have a runtime presence (e.g., processes, clients, database,and data stores) and trails of interaction e.g., communication links and protocols, information flows, use of resources, and access to common resources. Components are the major units of run-time interaction or data storage. Connectors are the interaction mechanisms among components. Different styles of Component and Connector views important vocabulary and often impose additional topological restrictions. In a shared data view, the data repository and the accesses are the components; the access mechanisms are the connectors. And in a client-server view, the components are clients and servers; the connectors are the protocol mechanisms by which they interact.

When considering documentation options it is important to be clear about criteria for choosing one way of using UML 2.0 over other possibilities. These criteria allow us to determine when a particular documentation option is likely to be appropriate. The criteria (the first three of which are derived from [8]) are

- Semantic match: The UML modeling elements should map naturally to the architectural concepts that are being documented.
- Visual clarity: The UML description should bring conceptual clarity to a system design, avoid visual disorder, and highlight key design information.
- Completeness: All relevant architectural features for the design should be implemented in the UML model.
- Tool support: Not all uses of UML are equally supported by all tools, particularly when specialize the UML.

The changes in UML 2 provide better representation options for many architectural concepts with respect to these criteria, particularly in terms of semanticoptions for architectural connectors remain deficient. Ideally, we look for a UML modeling option that is

- Typically used to represent trails of interaction,
- visually separate from component representations and introduces a minimum number of visual elements,
- able to represent connector behavior e.g., a state machine, state e.g., queue size, and interfaces over which a connector's rules might be imposed, and
- supported by UML tools implementing the minimal features needed to be compliant with the UML 2 standard.

3. DOCUMENTING COMPONENTS AND CONNECTORS WITH UML 2

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: components and component types, connectors and connector types, ports, roles, systems and properties [8].

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 [5].

3.1 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 packagable elements along with elements a class can own, which can be useful in some cases.

3.2 Documenting connectors

While the Connector element introduced in UML 2 is intended to represent communication links (a good semantic match to architectural connectors), it lacks the expressiveness needed to satisfy our completeness criteria. In particular, semantic information such as a connector's behavior or interfaces cannot be associated with a UML Connector. As a result, we examine other options available in UML 2 and how each compares to our criteria. We restrict our-selves to solutions that avoid importance of the UML to assist communication among varied documentation users, but briefly mention one simple specialization (examples of more complex specializations include [7, 9]).

Even though there is connector concept in UML, it lacks fluency 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 1).

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 1:- Documenting connector using link

The second strategy consists of using UML association class for describing connectors (Figure 2). 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.

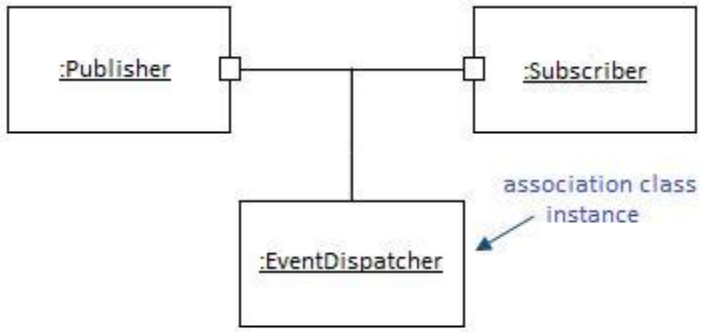

FIGURE 2:- Documenting connector using association class instance

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 3).

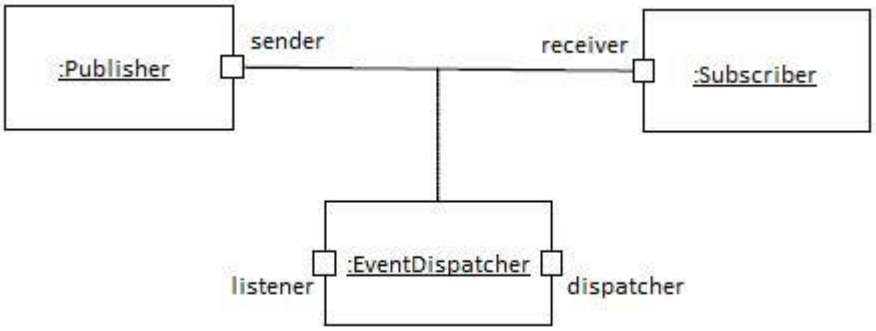

FIGURE 3:- Documenting connector using association class instance with ports

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 4).

FIGURE 4:- 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 5,

where UML components are used for representing architectural components, and class is used for representing connector.

FIGURE 5: - Documenting connector using object, UML components document components

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 rough 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. An Association, like a Connector or class connector, is a poor solution in terms of completeness. Because this type of stereotype require the graphical support which is not offered by the UML tools.

4. CONCLUSION

In previous versions, the UML has been an out of depth for documenting software architectures. Though architectural concepts could be represented using different UML modeling elements, in many cases there has been no clear best option for documenting some concepts, often resulting in compromising completeness or semantic match to fit within the UML vocabulary.

UML 2 is well defined tools for describing components and connectors from its predecessors. It has improved considerably by adding new things such as Structured Classifier and Port, which are excellent semantic matches to their corresponding architectural concepts. These improvements provide a clear best option, and reduce the confusion that can be caused by modeling the same architectural concepts differently for different systems or in different organizations.

However, there has not been a similar bound forward in terms of documenting architectural connectors. The new UML 2 Connector modeling element is not satisfactorily rich, making it difficult to associate detailed semantic descriptions or representations with them; consequently, they are a poor choice for representingcomponents and connectors.Also, concept of a part, introduced with composite structure diagrams in UML 2 can be taken into thought when showing complex properties of the containing structured classifier.

REFERENCES

1. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley (2001)

2. IEEE: 1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems (2000)

3. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Software 12 (1995) 42–50

4. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley (2000)

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

6. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Disci-pline. Prentice-Hall (1996)

7. Selic, B., Rumbaugh, J.: Using UML for Modeling Complex Real-Time Systems. White paper (1998)

Available at http://www.objectime.com/otl/technical.

8. Garlan, F., Cheng, S., Kompanek, A.: Reconciling the Needs of Architectural Description with Object Modeling Notations. In Evan, H., Kent, S., Selic, B., eds.: Science of Computer Programming, Special UML Edition. Volume 44. Elsevier Press (2002)

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

10. Group, O.M.: UML 2.0 Superstructure Specification: Final Adopted Specification (2003) OMG document ptc/08-03-02.

11. Ivers, J., Clements, P., Garlan, D., Nord, R., Schmerl, B., Silva, J.: Documenting Component and Connector Views with UML 2.0. Technical Report CMU/SEI-2004-TR-008, Sofware Engineering Institute, Carnegie Mellon University (2004)