A UML Profile for Documenting the Component-and-Connector Views of Software Architectures

Lic. Valerio Adrián Anacleto
Epidata Consulting
http://www.epidataconsulting.com
Buenos Aires, Argentina, Maipú 521 1ro A
adrian@epidataconsulting.com

ABSTRACT

In this paper, we present a UML profile and a group of UML patterns for documenting the component-and-connector views of software architectures [8]. They facilitate the creation of the component and connector viewtype in any UML 2.0 tool with a compliance level 3 [14]. This work's contributions are: (1) Facilitating the documentation of all the software application's views using only one tool. (2) Curtailing investment in personnel training. (3) Allowing the establishment of an adequate traceability between the architectural artifacts and the rest of the model.

Keywords: software architecture, component-and-connector viewtype, software documentation, UML 2.0

INTRODUCTION

The aim of this paper is to offer a solution, to a common challenge that arises in practice when an architecture is meant to be documented: how should the component-and-connector views of an application be documented in a syntactic and semantic correct way without losing the traceability of the rest of the documentation artifacts as well as using a unique documentation tool of software applications?

Until its 2.0 version, UML did not count with an appropriate support to document software architectures formally. However, since its 2.0 version, UML has added some new constructs such as: composite structures, ports and roles, which enable the architecture software documentation in a more natural and intuitive way. Although these constructs represent a clear improvement regarding the early UML versions, UML still falls short to document architectures formally [8] and even views as significant as the component-and-connector ones are not easy to document using UML [8].

Component-and-Connector Viewtype

The component-and-connector viewtype enables the representation of a software architecture from the point of view of its components, the principal unit of runtime interaction or data storage, and its connectors, the interaction mechanism among components and the "data flow" among them [8].

The component-and-connector view is considered one of the most important ones for the developer as well as for the architect [5] and of vital significance for the analysis and quality requirements scope, such as availability, performance, scalability among others.

When trying to design a component-and-connector view we come across a dilemma: should we model the component-and-connector view with an ordinary assistance tool for the UML design tool or should we use an architecture design tool such as: BiZZ design Architect [4], AcmeStudio [1], Aesop [2], Darwin [7] or Unicon [17]? For more information on ADLs and software architecture design tools, refer to [10].

Next, we will analyze some of the consequences of using diverse design tools for different documentation aspects.

Consequences of Using Different Tools

In everyday practice, the lack of formal knowledge of UML and software architecture as well as the need of books that wipe out the ambiguity in common errors using UML regarding the interpretation of design, analysis and documentation of applications in general already seems to be too much to force the use of multiple tools.

The use of a variety of tools implies a required training to use each tool, which is time consuming and that time means, in turn, an increase in the total cost of ownership (TCO) of the project.

It is feasible that the selected tools have a license cost, which implies a bigger investment in software.

Most companies own a UML design tool but they do not count with one that enables the design of software architectures.

The tools for software architecture design are neither established nor well known in the industry yet. Moreover, the best tools of this type are still only an academic initiative and, regretfully, the academic-industrial gap is big, and at the same time, the usability and quality of the tools are not the best.

Traceability of the documentation elements: for us, this is the most important of all the problems because of its impact on the usability and money expense as regards the documentation maintenance cost. By traceability we understand an existent relationship between the model elements. There can be different types of traceability relationships according to the particular requirements. For instance, the trace of the links among a diagram's elements or the trace that shows the evolution from the requirements to the final code, linking all the artifacts in between which, in a development process, usually represent the abstraction level and maturity of the solution in a specific stage of the development process. For instance, we could trace a group of requirements that gave life to a use case; at the same time, this use case could be traced with an analysis diagram and so on until we reach the code that implements the functionality of this use case. Having an appropriate traceability in a model allows, among other things, to analyze dependencies, to estimate the impact in the changes of an artifact, and to distribute the work and analyze the system's quality attributes by means of, for example, traces between the software components and the nodes where the deployment will take place.

By having two separate modeling tools for the architecture and for the rest of the application, the key traceability elements that create the links with the architecture are lost; therefore, they have to be kept separately, for instance, by means of using a traceability matrix.

Advantages of Using Different Tools

It is useful to use a variety of tools if, due to the characteristics of the application or the maturity degree of the software architecture practices, it is necessary to document that architecture using the highest possible level of detail and strictness.

Throughout a variety of software architecture assessments, we have discovered that, regardless of the tool and the type of view to document, it is imperative to count with a software architecture documentation that shows, among other things: the most important processes, the components that make up the architecture (sometimes called in industry "Architecture Map"), their dependencies and coupling, and in which abstraction level they occur (data, business logic, etc.).

It is striking that, in a quite high percentage, there are not many companies counting with a software architecture that meets these minimal conditions.

We believe that, in order to achieve this goal, it suffices to use any UML modeling tool in an adequate way, documenting the component-and-connector views within the same tool by means of a UML profile specification [14].

In this paper we present a UML profile by means of which the component-and-connector views in any UML modeling tool that supports a compliance level - complete (L3) can be documented, avoiding the consequences of using a variety of modeling tools.

The rest of the work is organized in the following way: in the first part we present our choice for modeling the component-and-connector views using UML 2.0 and representing it by means of UML patterns and profiles. In the second section we show a design, as an illustrative example, using the developed UML profile. Then, we analyze the work done from the point of view of usability. Finally, we make comments about some possibilities of future work and provide conclusions on the current paper.

MODELING THE COMPONENT-AND-CONNECTOR VIEWTYPE WITH UML 2.0

In order to facilitate the design of the component-and-connector views we have developed a UML profile [14] which, by being a standard specification defined by the OMG (Object Management Group), guarantees us that it will be able to be used by any tool that implements UML 2.0 or higher. The need to use the 2.0 or a higher UML version is due to the fact that in the 2.0 version some documentation constructs are introduced, such as ports and roles, which are useful for our UML profile.

Garlan's work [8] shows several of the available options for modeling a component-and-connector views using UML 2.0. We took as a basis the analysis done in that work for making decisions when creating our own UML profile. It is also important to mention that some of the decisions are original from this work; they will be described in detail later on.

Next, we will list the elements that make up the UML profile for the component-and-connector viewtype.

Components

We decided to document the components using the "Component" documentation construct defined in UML 2.0. Another option to document them was using the "Classes" documentation construct. There are some opinions that claim that the latter could vanish from the future UML versions since the semantics of both artifacts overlap considerably [12].

We decided to give the component a similar visual aspect to the one used in Christine Hofmeister's book [11] when documenting architecture diagrams, because we found it intuitive and practical for the purpose of documenting a software architecture.

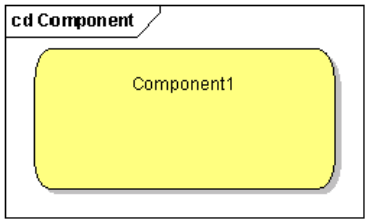

Figure 1: visual aspect of a component

Connectors

In all the papers and significant documents on software architecture documentation, the importance of treating connectors as "first class citizens" [15] is highlighted; in UML it could be expressed as a classifier and not as a simple association. The fact that a connector appears as a classifier has very deep implications in the expressive power of the connector and in the traceability of the artifacts; for example, one could create a connector as a structured classifier and within that connector define the class diagram related to the design itself, its sequences, collaborations and quality requirements specifications, which would greatly facilitate the analysis of quality

attributes that may or may not be reached by a software architecture.

In spite of the fact that we take Garlan's work [8] as a fundamental reference for our current work; the former does not include as an option the use of a "component" construct to document a connector. We believe that a connector's semantics is much more similar to the one of a component than to an association or an association class, which are the options mentioned in the quoted work.
We believe this because UML component are "first class citizen" [15] meanwhile association and association class aren't independent classifiers.

That is why in our UML profile we use as a basis a UML component to document a connector (from the component-and-connector viewtype). By changing its appearance by means of a stereotype, we can distinguish it visually from a component. We follow the visual aspect defined by Christine Hofmeister [11].

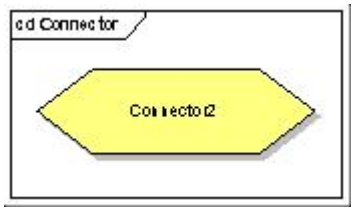

Figure 2: visual aspect of a connector

Ports
Ports are constructs defined in UML 2.0; therefore, we use them just as they are defined in the standard itself. Taking into account the fact that a port can only belong to a component, this restriction is not validated in the UML profile since not all the tools support OCL adequately.

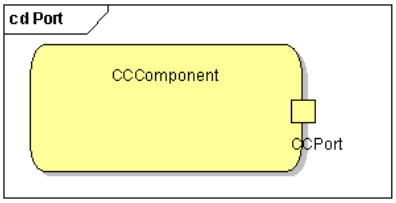

Figure 3: visual aspect of a port associated to a component

Roles
Just like in the case of ports, roles are defined in UML 2.0; therefore, we believe their use is quite convenient. However, a role will only be associated to a connector kind of construct.

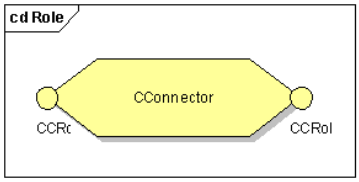

Figure 4: visual aspect of two roles associated to a connector

Association
To associate ports and roles, we use an association defined in our UML profile for the purpose of distinguishing it from other types of associations and allowing the subsequent development of tools that gain benefits and maximize the use of the current UML profile.

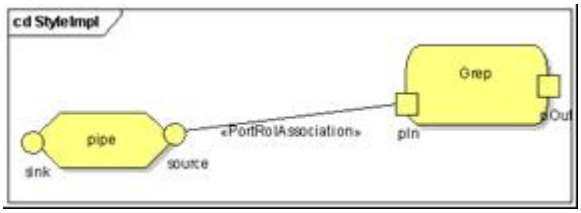

Figure 5: visual aspect and stereotype of the port-and-role association

Delegation
We thought it would be convenient to create a delegate association to distinguish it from the ordinary association. A delegate association only takes place between roles and it corresponds to the delegation of a message received by a port, which can be called "A" and which delegates the message to another one: port "B". In this case it could be said that port "A" delegates the message to port "B". We identify this association by placing an arrow on one of the ends of the line to indicate the message direction.

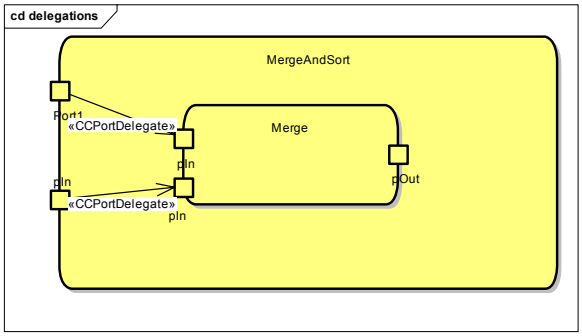

Figure 6: visual aspect and stereotype of two delegations

Properties
Our decision for documenting software architecture properties is to use tagged values, validated for the components' instances. In the case of properties shared by all the instances (type properties), we use attributes.

THE UML PROFILE

Next, we show the design diagram of the UML profile we have developed. It can be downloaded from [6]. It is important to mention that it was only used with the Enterprise Architect tool, which can be downloaded from [16]. Since a UML profile is a standard defined by the OMG, any tool that keeps to the standard should be capable of using the UML profile. Other tools that abide by the standard according to the OMG can be found in [18].

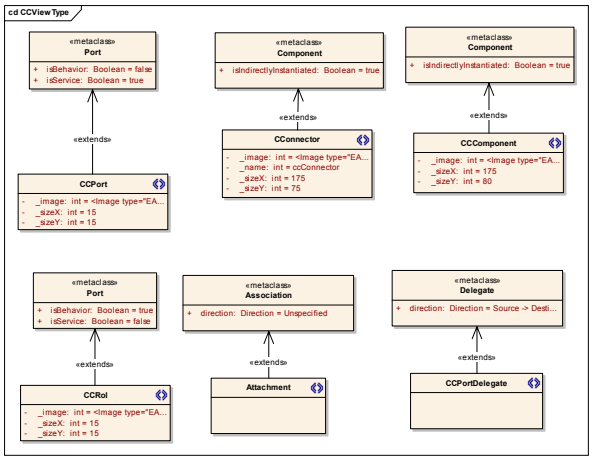

Figure 7: profile design

AN EXAMPLE ON HOW TO USE THE PROFILE

In figure 8 we show an example in which we document a pipe-and-filter sequence; the sequence itself is of no significance, but it is important to highlight the power of the developed profile. It represents instances of some components and connectors defined in figure 9.
The sample can be downloaded from [6]

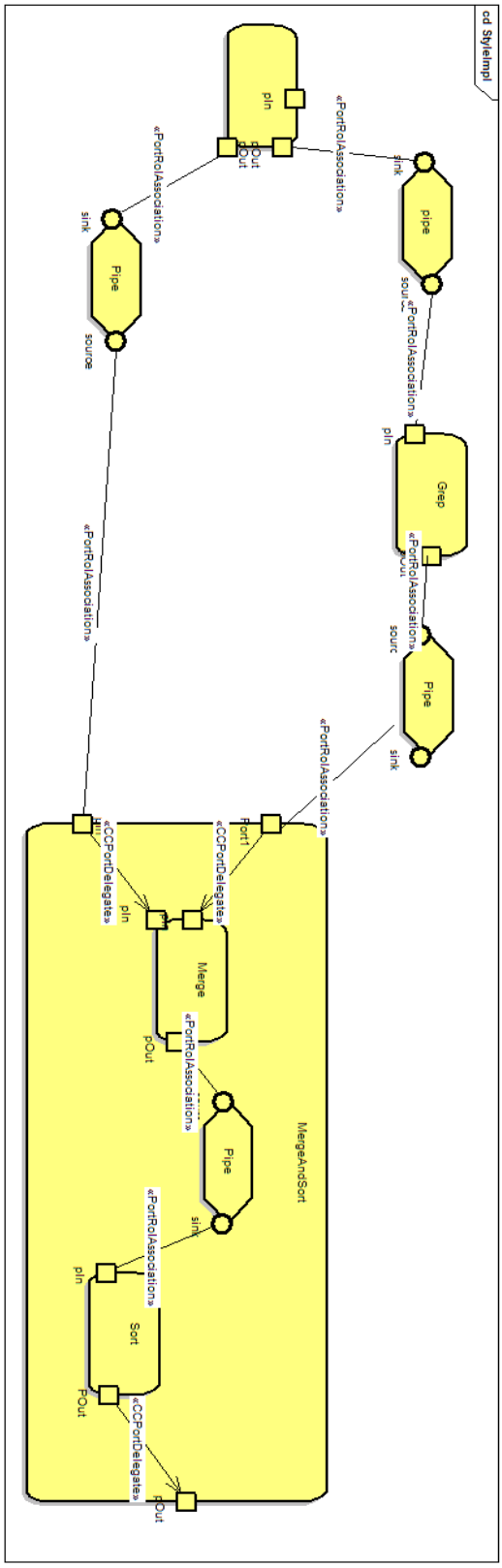

Figure 8: sequence of a pipe-and-filter view

The former diagram was created using instances of an architectural type diagram, which can be found in figure 9 and which represents the design of the components and connectors for a pipe-and-filter architecture style as well as their possible relationships, for instance, it can be noticed that a filter is related to a pipe through the ports of

the first and the roles of the second and that “grep”, “merge”, “sort” and “splitter” are specializations of “filter”, and as a consequence, they inherit its semantics.

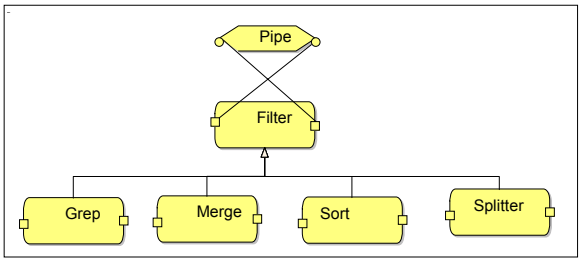

Figure 9: definition of architectural types for a pipe-and-filter architectural style

USABILITY

The International Standardization Organization (ISO) offers two definitions of usability:

ISO/IEC 9126: “Usability refers to the software’s ability to be understood, learned, used and considered attractive by the user, under specific conditions of use.”

This definition emphasizes the internal and external attributes of the product —regardless whether it is a software or not— which contribute to its functionality and efficiency. The usability depends not only on the product itself but also on the user. That is why a product is by no means usable by itself; it can only be used within a specific context and by specific users. The usability cannot be valued if a product is studied in an isolated way.

ISO/IEC 9241: "Usability is the efficiency and satisfaction with which a product enables specified users to achieve specified goals in a specified context of use”.

This definition focuses on the concept of quality of use, that is to say, it refers to the way the user performs specified tasks under specified circumstances effectively.

Usability of the UML Profile

It seems to be clear that, for the profile to be useful, it is necessary that it is usable in the sense given by both definitions above.

We implemented the profile bearing in mind the graphic aspects and following a well known component-and-connector metaphor taken from Hofmeister’s book [11], which bears resemblance to the one used in many works by Garlan, Shaw and other software architecture precursors and also to the iconography and metaphor used in some software architecture documentation tools, such as [17] and [1]. This makes the choice we have presented here intuitive.

There is a second aspect which has a deep impact in usability: it is that every component will have, at least, one port and every connector will have, at least, one role. The user will find it repetitive and tedious to drag and drop a port or a role every time he documents a component or a connector. To improve this aspect, we decided to use another standard defined within UML and implementing some component-and-connector design patterns. These patterns can be, for instance and among others, a component with a port, as shown in figure 3, a component with two ports, a connector with two roles, and so on.

This would considerably facilitate the use of the tool, making it more agile by avoiding repetitive tasks without losing the formality and semantics of components and connectors.

Integration of the UML Profile with Commercial Tools

In figure 10 we show the use of the UML profile with the Enterprise Architect tool. In the toolbox on the left, the menu is restricted for the component-and-connector viewtype with the elements defined in the UML profile. In the resource view on the right, the patterns that facilitate the use of the defined profile can be appreciated.
The profile integrated wth the tool can be downloaded from [6]

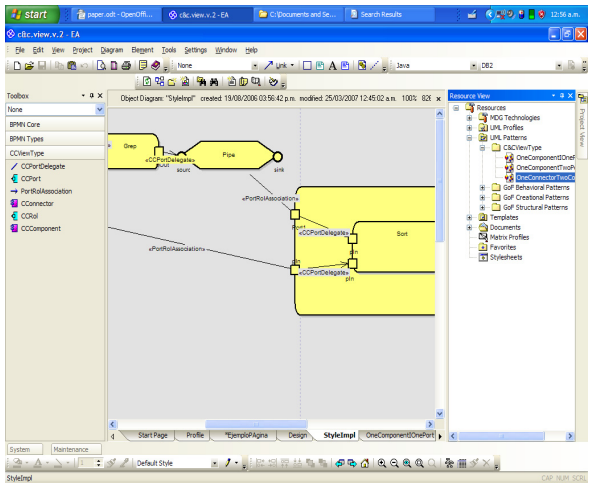

Figure 10: integration of the UML profile with commercial tools

FUTURE WORK

In future works, it would be interesting to look for the way to create traces between composite elements as if they were only one element, for example, a component and its ports. This would enable the traceability of a component with its ports and inner structure as a single element instead of many.

Integrating this profile with an ADL would be another possibility of future work, though it depends on OMG choosing an ADL for the standard UML, in case any decision is taken on the short run.

CONCLUSIONS AND CONTRIBUTIONS

In the current work, we implemented a UML profile for documenting the component-and-connector views. For the implementation of the profile, our main reference was Garlan’s work [8], though it was also necessary to take some decisions that were not taken into account in that work.

We believe that using this profile to document the component-and-connector views will facilitate the software architects’ work by enabling the documentation of the architecture together with the rest of the application’s design. By facilitating and, as a consequence, disseminating the use and practices of architecture, since a UML profile is a standard, it can be used with any UML design tool, diminishing the complexity of having to deal with a variety of tools and the overall cost of documenting a software application in an appropriate way.

REFERENCES

[1] Acme Project – home page: http://acme.able.cs.cmu.edu/index.html

[2] Aesop - home page: http://www.cs.cmu.edu/~able/aesop/aesop_home.html

[3] Len Bass, Paul Clements, Rick Kazman. Software Architecture In Practice, Second Edition. Boston: Addison-Wesley, p. 21-24. ISBN 0-321-15495-9, 2003.

[4] BiZZdesign Architect – home page: http://www.bizzdesign.nl/html/bizzdesignarchitect_en.html

[5] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford (2003). Documenting Software Architectures: Views and Beyond. Boston: Addison-Wesley, pp. 13-15. ISBN 0-201-70372-6.

[6] Component and Connector profile, patterns and sample downloads: http://www.epidataconsulting.com/tikiwiki/tiki-index.php?page=A+UML+Profile+for+Documenting+the+Component+and+Connector+Views+of+Software+Architectures+Downloads

[7] Darwin, The Software Architect's Assistant – home page: http://www.doc.ic.ac.uk/~kn/java/saaj.html

[8] David Garlan, James Ivers, Paul Clements, Robert Nord, Bradley Schmerl and Jaime Rodrigo Oviedo Silva. Documenting Component and Connector Views with UML 2.0. Technical report, CMU/SEI-2004-TR-008, Software Engineering Institute, 2004.

[9] David Garlan, R. Monroe, and D. Wile. "Acme: an Architecture Description Interchange Language," Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research, ACM, New York (1997), p. 7.

[10] David Garlan. Software Architecture: a Roadmap, in The Future of Software Engineering, A. Finkelstein, Ed.: ACM Press, pp. 93-101, 2000.

[11] Christine Hofmeister , Robert Nord , Dilip Soni. Applied software architecture, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999

[12] C. Kobryn. UML 3.0 and the Future of Modeling. Software System Modeling, 3:4--8, 2004.UML3.0

[13] MetaObject Facility (MOF) 2.0 Core Specification, Available Specification, OMG document ptc/04-10-15, Object Management Group (2004), http://www.omg.org/ cgi-bin/doc?ptc/2004-10-15.

[14] UML 2.0 Superstructure Specification, OMG document formal/05-07-04, Object Management Group, Inc. (2005), http://www.omg.org/cgi-bin/doc?formal/05-07-04

[15] Mary Shaw and David Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1995

[16] Sparx Systems - Enterprise Architect home page: http://www.sparxsystems.com.au/

[17] Unicon – home page: http://www.cs.cmu.edu/People/UniCon/

[18] UML Directory: http://uml-directory.omg.org/

Received: Jul. 2007. Accepted: Mar. 2008.