Integration of Composite Structure and Class Diagrams

Gregor v. Bochmann

Department of Electrical Engineering and Computer Science
University of Ottawa, Canada
bochmann@uottawa.ca

Abstract. In order to simplify the consistent integration of different system views, i.e. class diagrams, and composite structure diagrams, we propose that ports should be UML Components (in some sense), and connectors be associations. This entails that all these properties can be described with one type of diagram including classes, associations, components and composite structures, and the number of independent concepts in UML is reduced, thus simplifying the language. The paper first reviews the relevant UML concepts with a very simple example. Then the proposal is formulated by giving details about the correspondence between the features of ports and components, and connectors and associations, referring to the UML meta-model for this comparison.

1 Introduction

In the early times of UML, until around 2005, there was much discussion about how best to represent software architecture models in UML, in particular, how to represent the connections by which different software components are linked within an architecture [2, 3, 4]. While UML-1 had only limited facilities for representing such models, UML-2 introduced the new concepts of composite structure, ports and connectors which provide much better facilities for this purpose. Ivers et al [5] discuss several options for representing connections in software architectures using UML concepts, including UML connectors and associations. Bock [6] gives an informal explanation of the meaning of composite structure diagrams and also explains the meaning of connectors in terms of associations.

If one can explain the meaning of UML connectors through the meaning of UML associations, why is it necessary to introduce this new independent concept of "connector" in UML-2? – Would it not be better to have fewer concepts in this modeling language? After all, important criteria for language evaluation include simplicity and ease of learning.

The existing proliferation of concepts in UML also gives rise to difficulties regarding consistency between different views of the same system, for instance between class diagrams and composite structure diagrams. In order to simplify the consistent integration of system views defining properties of classes and associations with system views describing properties relating to the composite structure, we propose (in some sense) that ports are components, and connectors are associations (in the second point following the spirit of [6] – note that [7] also says "The natural choice for speci-

adfa, p. 1, 2011.

fying ADL connectors in UML is by stereotyping the standard UML Association element."). This entails that all these properties can be described with one type of diagram including classes, associations, components and composite structures.

In this paper, we first review in Section 2 the relevant concepts of UML-2 using a very simple example. In Section 3, we explain how ports can be considered as components, and how associations can be used to represent the connections in composite structures for which UML proposes the Connector concept. After an informal presentation of this proposal, we provide some rational arguments in its favor.

2 Review of some UML concepts and notations

Writing convention: In the following we write between " " the names that represent concepts in the UML meta-model [1]. We write in *italics* the names that are part of the example models discussed in this paper.

The **class diagram** in Figure 1 contains three "classes" *Company*, *Employee* and *Server*. It represents a universe of object instances consisting of companies, employees and servers. The two *aggregation* "associations" indicate that each *employee* has a single *employer* which is a *Company*, and each *server* has an *owner* which is also a *Company*. In addition, there is the *access* "association" which states that each *employee* may access some *servers* and each *server* may be accessed by several *employees*. The words *user* and *service* stand for the end-points of the association and represent the roles that the "linked" class instances play in the context of the association. Formally, each of these words is the "name" of a "memberEnd" of the *access* "association" which is in fact a "property" of the linked class instance (see [1], Figure 7.12). A "property" of a "class" is either an attribute (such as the *name* of the *Employee*), the end-point of an "association" (as in the case discussed here), or a "port" or some other property. We note that one sometimes uses a similar notation for association end-points as for attributes, for instance, the *access* association could be represented as an attribute of *Employee* written as "service: Server [*]".

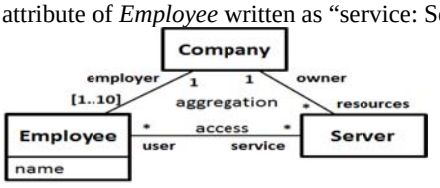

Fig. 1. A Class diagram

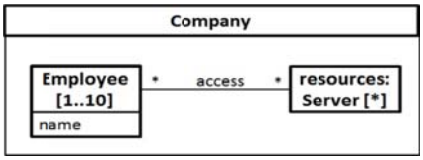

Fig. 2: Corresponding composite structure diagram

It is important to note that in the model of Figure 1, the access association may be such that an employee may access a server where the owner of the server is different from the employer of the employee. If that is not desirable, we may consider the following constraint:

Server-Protection: *An employee can only access servers that are owned by his employer.* - - This constraint can be expressed in OCL as follows:

context access : self.user.employer = self.service.owner

Composite Structures

Figure 2 shows a UML Composite Structure diagram where the company "class" contains two kinds of component classes: *Employee* and *Server*. The multiplicities are the same as in Figure 1. The line between the two component classes is a "connector" which has the same "name" and multiplicities as the *access* "association" in Figure 1. The semantics is the same as the Class diagram of Figure 1 including the *Server-Protection* constraint defined above. As explained in Section 9.3.13 of [1], the class instances depicted in a Composite Structure diagram within a given composite class represent only those instances that are related to a particular instance of the composite class - in Figure 2, to a particular instance of Company. Note: We ignore in this paper the distinction between composition and aggregation relationships for the composite class with its components, as shown in Figure 9.20 of [1].

In the meta-model of UML, a "connector" has two "connectorEnds". Each "connectorEnd" has a "role" that points to "connectableElement" which is normally a "class", such as Employee or Server in our example. A "connectorEnd" may also have a "partWithPort" which points to a "port".

The concept of a "port" is useful when one wants to describe objects that have several interaction points through which different kinds of interactions take place. Figure 3(a) is an extension of Figure 2 where a server interacts not only with the users, but also with the support staff. This diagram does not use ports since the details of the interactions are not specified. Figure 3(b) shows the same situation, and two "ports" are distinguished for the server instances. The interactions through the user port are different than the interactions through the staff port. In general, a "port" represents two kinds of "interfaces": "provided" and "required". An "interface" defines "attributes" and "operations". By default, a "class" has a "provided" interface that defines the visible attributes of the class instances and the operations that may be invoked on an instance. Similarly, the attributes and operations defined by a "provided" interface of a "port" will be made available by the object instance to which the port belongs. On the other hand, it is assumed that this object instance will require having access (through the same port) to the attributes and operations defined by the "required" interface of the port. In Figure 3(b), the *staff* port of the servers is associated with a "port type" *MaintenancePort* (using the notation shown in [1], Figure 9.16) and the port type of the corresponding port of the support staff has the "conjugated" port type (indicated by the "~" symbol) which means that the "provided" and "required" interfaces are interchanged. In our example, the *MaintenancePort* port type could have a "provided" interface that provides access to an attribute *CPUutilisation* and an operation *performanceTest*, and a "required" interface that supports the execution of an *alarm* operation.

Figure 3(c) shows a situation similar to Figure 3(b), except that the support staff is not part of the company. Here the company has a port m of type MaintenancePort through which the support staff has access to the servers of the company. The "connector" between the staff port of the servers and the m port of the company is a "connector" of kind "delegation", since the requests arriving on the m port will be delegat-

ed to the staff port, and inversely. The other connectors on this diagram are of kind "assembly".

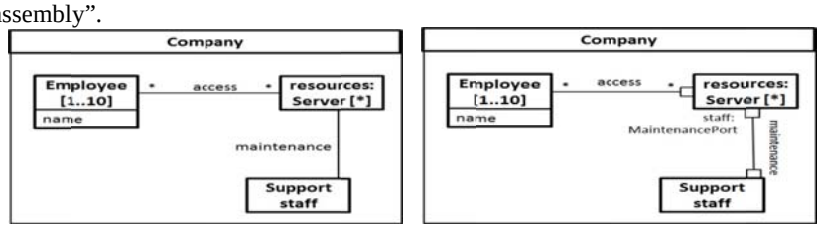

Fig. 3. Composite structure including support staff – (a) without ports - (b) with ports. Figure 3(b) may be considered a refinement of Figure 3(a).

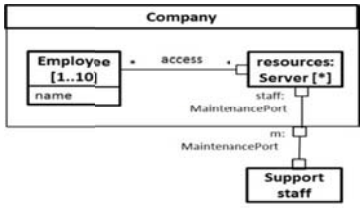

Fig. 3. (c): with external support staff.

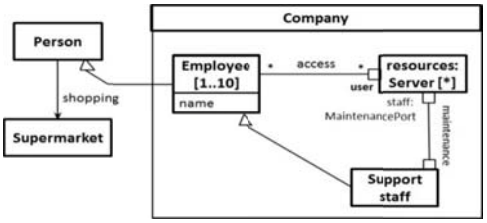

Fig. 4: Class diagram with composite structures

3 Integration of component structure with class diagrams

We propose an integration of the concepts in UML Class diagrams and UML Component Structure diagrams by reducing the number of different concepts involved. Specifically, we propose the following: (a) To consider that ports are components. (b) To use the concept of associations instead of connectors for defining the interconnections in composite structures. (c) To extend UML Class diagrams to include classes with ports and hierarchical structures for aggregation or composition. In principle, such a proposal requires a revision of the definition of UML and its meta-model. Since this would be a difficult undertaking, we take here an informal approach. We propose:

1. Ports are components, and as such have provided and required interfaces. If a behavior is specified for such a component, it represents a specification (sometimes called "protocol") that defines constraints on the order of interactions occurring at its interfaces. The realization of a conforming behavior is left to the component to which the port belongs.
2. The instance names of components within a composite structure are the same as the corresponding "memberEnd" of the implicit aggregation association between the component and the composite classifier. See, for instance, the name "resources" in Figures 1 and 2.
3. Connectors within a hierarchical structure are represented by special kinds of associations, where the endpoints of an association correspond to the "ConnectorEnds" of the corresponding connector. Associations between ports may be of kind "as-

sembly" or "delegation", and one may also distinguish between operation call or message transmission associations. As naming conventions for these associations, we suggest the following.

(a) The name of the association is the same as the connector, if any.
(b) For "assembly" connectors, the names of the association ends are the same as the names of the connected ports. Note, however, that these names change their position in the hierarchical structure diagram – the port name is located close to the port symbol, while the name of the association end would normally be located at the other end of the association. If port names are present, there is no need for the names of the association ends. Compare Figure 1 and 4.
(c) For "delegation" connectors, the corresponding association between the ports is in a sense a refinement of the implicit aggregation association between the composite structure and the contained component.
(d) As usual, interfaces of ports can be shown in diagrams as lollipops.

4. Following the points (1) to (3) above, mixed UML diagrams, such as Figure 4 (including classes with hierarchical composite structures and components with ports) obtain a clear semantics. Note that such mixed diagrams are already allowed according to UML (see [1], page 694).

The following are some rational arguments for this proposal.

A port is a component. Considering a "port" to be a kind of "component" can be justified by the following similarities between these two concepts:

1. A port is a part of a StructuredClassifier ([1], see "ownedPort" in Fig. 9.4) and similarly components are "part" of StructuredClassifiers (see [1], Fig. 9.2).
2. Ports and components have provided and required "interfaces" ([1], Figs. 9.4, 8.2)
3. Both can be associated with a dynamic behavior that imposes constraints on the order of interactions at the associated interfaces.
4. In UML, both concepts, "ports" and "components", can be linked by "connectors".
5. Note that "port" is not a "classifier" in UML. Making it a "component" (which is a "classifier") allows ports to be linked by associations.

Associations are used to link parts of a Structured Classifier. Using an "association" to represent a connector between the components within a software architecture has been considered by several authors (see for instance [6]). This approach can be justified by the following similarities between these two concepts:

1. An association has two or more endpoints which are "properties" of some "class" (see "memberEnd" in [1] Fig. 7.12). Similarly, a "connector" has two "connectorEnds", each having a "withPort" which also is a "property" of some "class" or has a "role" which is usually a "component" (see [1] Fig. 8.3).
2. An "association" is used to link "classes". Since "components" are "classes" they can be linked by "associations. If ports are components (see above) then also ports can be linked by "associations".

One of the issues discussed in [5] was the representation of connections that have more complex behavior, such as message loss or transmission delays, or other behav-

iors, and the use of three-way associations or classes was proposed for this purpose. We think that it is more natural to represent such a connection by a class with two ports that would be linked to the two endpoints of the original connection.

We also note that a single port may be connected (associated) to ports of different components thus allowing the routing of messages or operation calls to the appropriate component (as supported by the channels in SDL [8]).

4 Conclusions

We have proposed in Section 3.1 changes to UML in which ports would become components and connectors would be associations, given the similarity of their semantics. This has the advantage that UML becomes simpler, since the number of independent concepts is reduced, and that the consistency between different viewpoints, such as expressed by UML Class and CompositeStructure diagrams, is easier to establish. We plan to integrate this approach into an extension of Umple [9] for supporting composite structures.

Another question of simplification, not discussed in this paper, is the similarity between UML Components and StructuredClassifiers. A component is a black-box view of a Classifier, however, it also needs a behavior specification. A component is similar to an SDL block [8] which can be refined by providing a state machine behavior or some hierarchical sub-component structure.

Acknowledgements: We thank Dorina Petriu and Tim Lethbridge for their comments

References

1. OMG Unified Modeling Language (OMG UML), Superstructure UML, Version 2.4.1, http://www.omg.org/spec/UML/2.4.1/
2. C. Kobryn, Modeling components and frameworks with UML, Communications of the ACM, Vol. 43, 10, Oct 2000, pp. 31-38.
3. D. Garlan, A. J. Kompanek, Reconciling the needs of architectural description with object-modeling notations, in Proc. Int. Conf. UML 2000, Springer LNCS (2003), pp. 498-512.
4. C. Kobryn, Modeling components and frameworks with UML, Communications of the ACM, Vol. 43, 10, Oct 2000, pp. 31-38.
5. 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)
6. C. Bock: "UML 2 Composition Model", in Journal of Object Technology, vol. 3, no. 10, 2004, pp. 47-73, http://www.jot.fm/issues/issue_2004_11/column5
7. V. Issarny, A. Zarras, Software architecture and dependability, in Formal Methods for Software Architctures, Springer LNS 2804 (2003), pp. 259-286.
8. Specification and Description Language (SDL), see for instance http://www.sdl-forum.org/SDL/Overview_of_SDL.pdf
9. M Garzon, H Aljamaan, TC Lethbridge, Umple: A Framework for Model Driven Development of Object-Oriented Systems, in Proc. of Software Analysis, Evolution and Reengineering (SANER), 2015, IEEE, pp. 494-498. See also http://cruise.eecs.uottawa.ca/umple/