Modeling Crosscutting Concerns using Software Connectors

Mohamed Mancona Kandé and Alfred Strohmeier

Swiss Federal Institute of Technology Lausanne (EPFL)
Software Engineering Laboratory
CH-1015 Lausanne EPFL, Switzerland
Email: {mohamed.kande, alfred.strohmeier}@epfl.ch

1. Introduction

As communication between components (entire software systems) becomes increasingly important, the complexity of software connectors varies widely from primitive units of associations supported by programming languages, such as procedure calls, to sophisticated communication mechanisms, such as those supported by middleware platforms. Capturing various kinds of *architectural concerns* that crosscut the boundaries of individual software components and reasoning about their properties is a non-trivial task which needs to be tackled.

To address these issues, researchers in the field of software architecture have advocated the concept of a software connector as a critical element in modeling software architectures. A number of architecture description languages (ADLs) and tools that support those languages have been developed [13][8] [10].

In [13], Garlan and Shaw point out that a connector, like a component, requires a specification that characterizes its properties in various ways. They extensively argue for the connector concept as a first-class modeling element that should be defined in any ADL. According to [13], *connectors* mediate interactions among components; that is, they establish the rules that govern component interaction and auxiliary mechanisms required. With this definition, the authors apply the principle of separation of concerns in an advanced form. They define software components and their interconnections (connectors) as two separate architectural abstractions that are orthogonal to each other. We refer to this as a way of using *advanced separation of concerns* [16] [17][18] in software architecture.

As a result, an architect should be able to separate the identification, representation and reasoning about software concerns that can be localized on a single component; from those that crosscut the boundaries of individual components. Unfortunately, the software architecture community has not defined the exact nature of connectors [9].

Very recently, Medvidovic and his colleagues [9] have proposed a classification framework based on the above definition, which aims at providing a taxonomy of connectors. The taxonomy enriches the set of requirements defined in [13], and it shows relationships among multiple kinds of software connectors and classifies them in service categories, connector types, dimensions and values. *Service categories* represent the interaction roles the connector fulfills, such as facilitation, coordination, and communication. *Connector types* distinguish different realizations of connectors, such as, procedure call, event and data access. *Dimensions* capture various kinds of details related to a connector type. For instance, the event connector type can have the dimensions delivery, priority, synchronicity, etc. A dimension can be composed of other dimensions, called *sub-dimensions*, and the taxonomy defines several sub-dimensions for the delivery dimension, such as: best effort, exactly one, at least one, etc. Lastly, each dimension and sub-dimension can take one or more *values*.

This taxonomy facilitates better understanding of software connectors by classifying results of many research projects in an easy to understand framework. However, it does not take into account how mechanisms of advanced separation of concerns should drive software architecture description, including connector modeling.

This position paper is motivated by work on architectural modeling with UML [15] which has exposed the need for connectors that support the definition of complex transactions and that provide mechanisms for maintaining information about the state of these transactions. Also, as discussed in previous work [4], one purpose of this paper is to show, by a concrete example, how an extended UML can be used to model a complex software connector. We believe that this is important, since the standard UML [11] does not allow one to specify “simple connectors” (interconnections between two parts) as a separate model element that can stand alone without the parts it interconnects; nor does UML provide support for modularizing complex component interactions. For example, model elements such as associations, links, dependencies and communication relationships cannot exist alone in a UML model.

The next section introduces the ConcernBASE[1] approach [19] to connector modeling that adapts the classification framework described above. We started investigating the ConcernBASE approach two years ago with two separate objectives: (1) definition of "UML Profiles" that provide explicit support for architecture description, taking into account the multidimensional nature of software architecture; (2) development of a UML-based tool that supports concern-based modeling using an architecture-centered software develop-

1.ConcernBASE stands for ***Concern-B***ased and ***A***rchitecture-centered ***S***oftware ***E***ngineering ment method.

2. Connectors Modeling with ConcernBASE

This section briefly presents the ConcernBASE approach to connector modeling, using some extended UML notation and the principle of multidimensional separation of concerns (MDSOC) [14]. In particular, it emphasizes support of architectural concerns that crosscut the boundaries of single components.

To help understand this approach, we apply it on the case study described as follows: We consider an online auction system, which enables multiple, geographically distributed users to participate, simultaneously, in various auctions via the Internet (we use here the English Auction style only). Each auction has a duration that is fixed in advance. The software allows users to subscribe to the system for bidding on and selling of goods. Successful subscription means that the user becomes a customer of the system. A customer can browse the list of current auctions, increase the credit of his/her account with a certain amount of money, join an existing auction or start a new auction. Whenever a customer wants to start an auction, s/he must indicate the name of the auction and the opening price (the minimum, acceptable price for the proposed item). A customer can participate in any ongoing auction as a buyer, but s/he must be registered in that auction. A customer can log off and log into the system whenever s/he wants. When logging in, a user is automatically informed of the status of the auctions that s/he is involved in. An auction closes when the specified auction period expires. The goods are only sold if at least one valid bid has been placed before the auction ends. At the end of an auction, the participant that has placed the highest bid is declared the winner and the amount of money corresponding to the value of the highest bid is transferred from the winner's account to the seller's account minus the commission charged by the auctioning body.

The software must also meet a number of other quality attributes or non-functional requirements, such as, concurrent use of the system, reliability, persistence, and fault tolerance.

The approach consists of the following steps:

- Define a concern space for connector modeling.
- Create different architectural views on software connectors.
- Describe the connector along the dimensions of the concern.

2.1. Define a Concern Space for Connector Modeling

Using MDSOC, we regard a connector as a *sphere* of interactions among software components. The connector construct that we propose aims at providing capabilities that allow one to define a context for various kinds of *interactions*, simultaneously. To capture the *interaction concerns* embodied by the connector, we define a *structural concern space* for connector modeling that adapts the previously outlined taxonomy. For this purpose, we *identify* all concerns of importance for connector modeling, *encapsulate* the concerns using units (UML elements or extensions), identify and manage *relationships* between the concerns, and *integrate* the concerns.

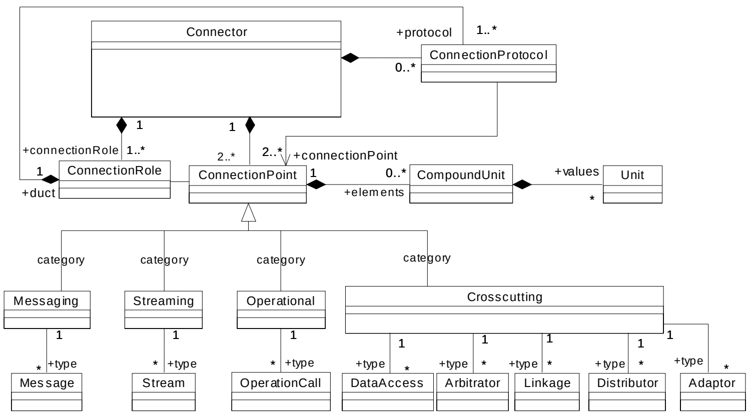

Figure 1. Structural Concern Space for Software Connector Modeling

Figure 1 shows the structure of the space of connector concerns. It describes a UML-based instantiation of the classification framework for software connectors. The connector concern space organizes all the important concerns that are required to model software connectors. Although not shown here, this model distinguishes between *simple connectors*, and *complex connectors* (also referred to as *higher-level connectors*) [4]. A simple connector type consists of one *connection role* and *two connection points*. Each connection point is characterized by the category of interaction service it supports. Interaction services involve different kinds of communications, such as *streaming*, *operational*, and *messaging*. The types of connectors supported by these three categories are *streams*, *operational calls* (including object-oriented method and procedure calls) and *messages*, respectively. A feature of these categories is that when a component sends a stream, message or a call event to another component, the receiver will have to perform a certain computation, according to its specification. The computation the component performs is localized within its boundary. In contrast, a common characteristic of crosscutting interaction services is that their behavior usually crosscuts the boundaries of individual components. The connector types supported by these categories are *data access*, *arbitrator*, *linkage*, *distributor* and *adaptor*, which are defined in the connector taxonomy. Data access connectors might be used for communications between components (providing access to data) or for transforming data being accessed from one form into another when crossing the boundaries of multiple components (conversion).

A connection role is an abstract representation of a channel that carries information exchanged between two interacting components and links the connection points. A connection point represents the connector interface. It defines the place at which a component joins a connector to interact with another component.

A higher-level connector type is a composition of two or more simple connector types. In addition to a set of simple connector types, the specification of a higher-level connector contains zero or more *connection protocols*. These additional protocols might be used to link connection points of different simple connectors in new compositions. A connection protocol specifies a communication pattern, which defines the ordering of the flow of information and control between connection points.

In this instantiation of the classification framework, we decided not to use the notion of dimensions and subdimensions. Instead, we use *compound units* and *primitive units*, which can be composite or simple UML model elements.

So far, the concern space is able to identify the concerns that are significant to a connector and to show their mutual relationships. Later on, we will discuss the UML extensions that are needed to support this approach. While some of the required model elements have already been defined in the context of a UML profile for architectural modeling [4], many of them need to be refined in order to provide better support for advanced separation of concerns. The resulting connector is in fact a multidimensional construct, i.e., an architectural modeling concept that provides capabilities for identifying, encapsulating and integrating multiple kinds (dimensions) of *interaction concerns* using extensions of UML.

2.2. Create different architectural views on software connectors

To understand and specify different aspects of a multidimensional concept of software architecture, different architectural views are required. This enables one to focus on different aspects of software connectors. In this section, we create different architectural views of the connector model, which describe the static structure, dynamic structure and configuration structure of the online auction system. These structures will be described by the following architectural views: *static*, *behavioral*, and *configuration*. Each architectural view represents a projection of a part of the structural concern space from a particular perspective or viewpoint [5].

2.3. Describe the connector along the dimensions of the concern

This section describes how to specify various kinds of component interaction concerns that pertain to the static, behavioral, and configuration views using the Auction case study.

Static View of the Auction system

The static view describes the static structure of the connector model, using the example of the online auction system. The auction system is transaction-based system that can be regarded as a complex, structured collaboration. As a transaction-based system, the auction system presents many features, such as atomicity and persistency, which are crosscutting concerns. To allow modeling such concerns in UML, we extend the notion of connection points, which was originally defined in [4], by making a clear distinction between interaction points and connection points. Interaction points are part of the interface of a component whereas connection points are part of the interface of a connector.

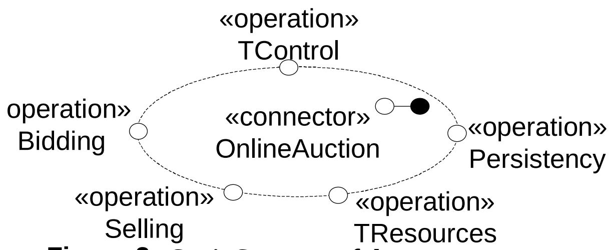

Figure 2. Static Structure of the OnlineAuction Connector

Figure 2 shows the static structure of the OnlineAuction connector. It consists of five different concerns: concurrency control, persistence, resource management, selling goods, and bidding for goods. The concurrency control concern involves both coordination and facilitation services. The type of connector needed to represent it is an arbitrator connector because it arbitrates concurrent accesses to resources. The language element used to describe it is the Tcontrol connection point. Along similar lines of reasoning, we can derive the following concern/connection points pairs: (persistence, Persistency), (resource management, TResources), (selling goods, Selling), and (bidding for goods, Bidding).

Figure 3 shows the details of some of the connection points of the OnlineAuction Connector. Note that these are not to be confused with simple, which make use of both compartments to define two connection points that are conjugates of each other.

Behavioral View of the Auction system

This architectural view emphasizes the behavioral aspects of the OnlineAuction Connector. It allows one to better understand and describe the crosscutting struc-

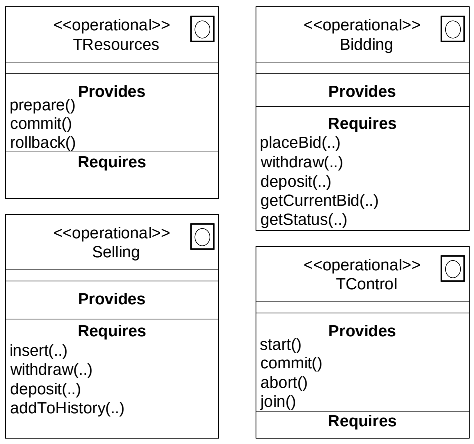

Figure 3. Connections Points for the

tures and the dynamic aspects of component interactions. Figure 4 shows a scenario in which three activities crosscut the boundaries of several components. Jim (the seller) using the TControl connection point starts the auction, resulting in a new transaction, T1, and a new auction is created (through the TResources connection point) then inserted into the list of currentAuctions (Selling connection point). At a certain point in time, another participant, John, joins the auction and consequently the transaction T1. The action of placing a bid (Bidding connection point) starts a new (sub-) transaction T1.1. From within the sub-transaction T1.1, the bid amount is withdrawn from John's account (Selling connection point). Since he has sufficient funds the withdrawal is successful and it is announced to the other participants. Another participant, Jack, decides to make a bid with the system and consequently a new sub-transaction is created, T1.2. The new bid is announced, consequently T1.1 is aborted (TControl connection point). The withdrawal from John's account is rolled back. Jack is declared the winner of the auction.

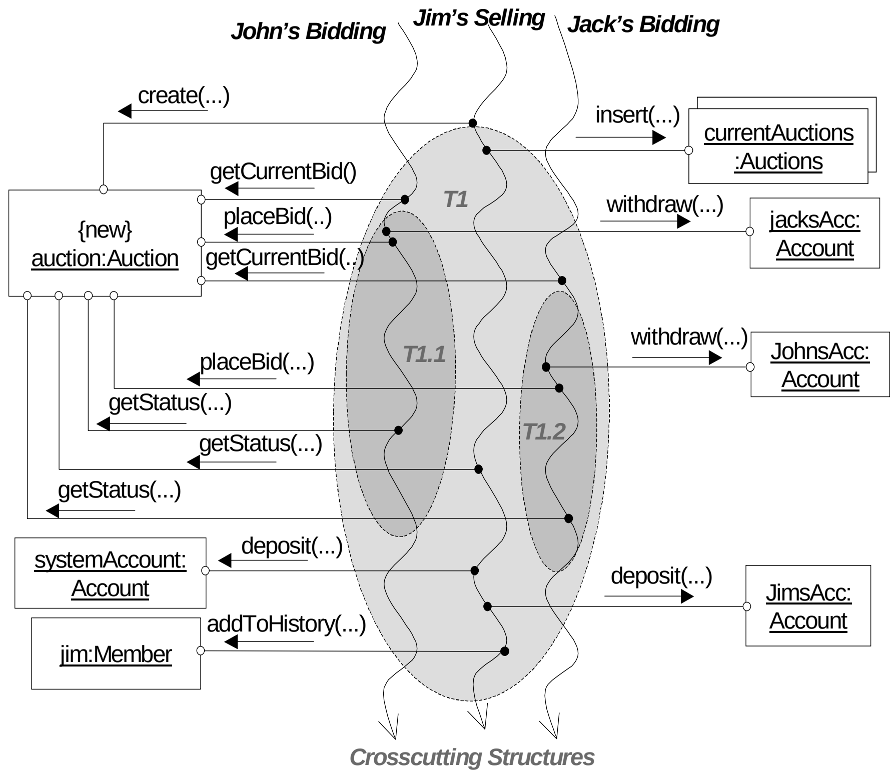

Figure 4. Behavioral Structure for the OnlineAuction Connector

Figure 4 shows the calls made by the three participants, where the black dots indicate the calling points and the white dots indicate the reception points. This illustrates the idea behind connection points pairs, conjugates of each other. The consistency between the static view and the behavioral view can be fulfilled by mapping a set of calling and reception points to corresponding connection points, which in turn are grouped to form the interface of the connector. This is a powerful mechanism that allows us to distinguish between the interface of the components (interaction points) and the interface of the connector (connection points). Therefore, it allows us to separate calls, as defined in connection points, from signatures of operations, as defined in interaction points.

Configuration View of the Auction system

The configuration view describes the decomposition of a system in terms of component and connector instances and the corresponding constraints on and between them. It specifies the rules and guidelines for creating instances of connection points and attaching them to the components.

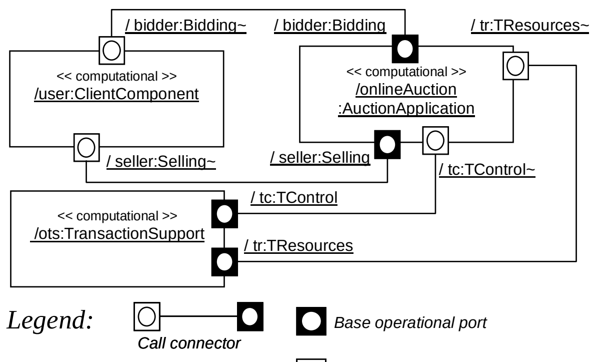

Figure 5. Configuration View for the Auction System

Figure 5 shows three component instances: user, onlineAuction, and ots. It illustrates a possible instantiation of the OnlineAuction connector and its composition. This view describes the organization of the system in terms of component instances that are interconnected by an instantiation of the OnlineAuction connector. This example shows an instantiation of each connection point defined in Figure 5 and the way they are plugged into the component instances. Simple connectors consist of two ports, which are dynamically created and plugged into the interacting components; they are conjugates of each other. Conjugated ports are shown with the tilde symbol '~'.

3. Summary

The main contribution of this paper was a proposal for a connector model that provides UML support for modeling a sphere of component interactions, and that takes into account concerns that crosscut the boundaries of individual architectural software components. The paper applied our ideas to an auction case study, and showed how our proposed connector model helped to understand some issues about architectural concern modeling. In our approach, we advocated advanced separation of concerns that, we believe, will help deal with many architectural modeling issues in general, and particularly in the context of UML.

4. References

[1] Allen R. *A Formal Approach to Software Architecture*. Ph.D. Thesis, Carnegie Mellon University, School of Computer Science, available as TR# CMU-CS-97-144, May (1997).

[2] Bass L., Clements P., and Kazman R. *Software Architecture in Practice*. Addison-Wesley 1998.

[3] Garlan D., Monroe R., and Wile D. *ACME: An Architecture Description Interchange Language*. Proceedings of CASCON '97 (1997).

[4] Kande M., Strohmeier, A. *Towards an UML Profile for Software Architecture Descriptions*. UML'2000 - The Unified Modeling Language: Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Kent, S., Evans, A., Selic, B. (Ed.), LNCS (Lecture Notes in Computer Science)

[5] Kande M., Strohmeier, A. *On The Role of Multi-Dimensional Separation of Concerns in Software Architecture*. Position paper for the OOPSLA'2000 Workshop on Advanced Separation of Concerns. (On-line at http://lglwww.epfl.ch/~kande/Publications/role-of-mdsoc-in-swa.pdf)

[6] Kienzle J., Romanovsky A., and Strohmeier A. Open Multithreaded Transactions: Keeping Threads and Exceptions under Control. 6th International Workshop on Object-Oriented Real-Time Dependable Systems, Italy, January 2001.

[7] Kienzle J. *Open Multithreaded Transactions: A Transaction Model for Concurrent Object-Oriented Programming*. Ph.D. Thesis EPFL-DI, no 2393, Swiss Federal Institute of Technology in Lausanne, Software Engineering Lab., 2001.

[8] Medvidovic N., and Taylor R. *A Classification and Comparison Framework for Software Architecture Description Languages*. IEEE Transactions on Software Engineering, Vol. 26, No.1, January 2000.

[9] Mehta N., Medvidovic N., and Phadke S. *Towards a Taxonomy of Software Connectors*. Proceedings of the International Conference on Software Engineering - ICSE'00 (2000).

[10] Moriconi M., Riemenschneider R. *Introduction to SADL 1.0*. SRI Computer Science Laboratory Technical Report SRI-CSL-97-01, March 1997.

[11] OMG Unified Modeling Language Revision Task Force. *OMG Unified Modeling Language Specification*. Version 1.4 draft, February 2001. http://www.celigent.com/omg/umlrtf/

[12] Selic B., Gullekson G., and Ward, P. *Real-Time Object-Oriented Modeling*. Wiley, 1994.

[13] Shaw M., and Garlan, D. *Software Architecture - Perspectives on an Emerging Discipline*. Prentice-Hall, New Jersey (1996).

[14] P. Tarr and H. Ossher. *Multi-Dimensional Separation of Concerns and The Hyperspace Approach*. In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2000. (To appear.)

[15] Weigert O. (moderator). *Panel: Modeling of Architectures with UML*. In UML 2000 — The Unified Modeling Language: Advancing the Standard, Third International Conference, S. Kent and A. Evans (Ed.), LNCS, York, UK, October 2-6, 2000.

[16] Tarr P., Ossher H., Harrison W., and Sutton S. Jr. *N-Degrees of Separation: Multi-Dimensional Separation of Concerns*. Proceedings of the Interna-

tional Conference on Software Engineering - ICSE'99 (May 1999).

[17] S. Clarke, W. Harrison, H. Ossher, P. Tarr. *Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code*. In proceedings of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) Denver, Colorado U.S., November 1999.

[18] G. Kiczales et al. Aspect-Oriented Programming. In ECOOP'97 proceedings. Finland.

[19] V. Crettaz, M. M. Kandé, S. Sendall and A. Strohmeier. *Integrating the ConcernBASE Approach with SADL*. To appear in UML 2001 Proc. Martin Gogolla (Ed.), LNCS (Lecture Notes in Computer Science).