Towards a Periodic Table of Connectors

Dan Hirsch, Sebastián Uchitel, and Daniel Yankelevich *

Departamento de Computación, Universidad de Buenos Aires,
Ciudad Universitaria, Pab.I, (1428), Buenos Aires, Argentina
{dhirsch, biuti, dany}@dc.uba.ar

The software architecture of a system defines its high level structure, exposing its gross organization as a collection of interacting components. Connectors are the glue for combining components and are a critical aspect of architectural design. Thus it is important to consider connectors as first class entities and provide simple, yet abstract descriptions of them. Although many formalisms model connectors explicitly, their descriptions are given in an operational manner capturing low level properties that usually only model communication patterns. If connectors could be characterized by means of high level properties, descriptions would be much more declarative and adequate for architectural analysis. Moreover, if these high level properties were thought of as primitives, a systematic analysis of connectors could be done. A canonical set of primitives would provide a framework for comparing, refining and reusing connectors. Consider two connectors characterized in terms of primitive properties, if a common subset of properties exists, a new type of connector could be factorized. On the other hand, if there is no property that allows both connectors to be differentiated, then the existing set of canonical properties must be refined. Finally, such a framework allows connector universe exploration, by contriving connectors as new combinations of properties.
In this paper we develop a framework for connector modeling following these ideas and put forward a first sketch of primitive connector properties. We also show how this framework can be used to define operations and reason about connectors.

1 Introduction

The software architecture of a system defines its high level structure, exposing its gross organization as a collection of interacting components [3]. Connectors are the glue for combining components and are a critical aspect of architectural design. Thus, it is important to consider connectors as first class entities and provide simple, yet abstract descriptions of them [1]. Although many architecture description languages such as Wright [2], Unicon [7] and ACME [3], model connectors explicitly, their descriptions are given in an operational manner capturing low level properties that usually only model communication patterns. In

The authors were partially supported by ARTE Project, PIC 11-00000-01856, ANPCyT y TW72, UBACyT.

Wright for instance, describing a connector requires the specification of connector roles and a CSP-like notation, called glue, for describing role interaction.

If connectors could be characterized by means of high level properties, descriptions would be much more declarative and adequate for architectural analysis. For example, it is important to identify connectors that support request/reply interactions and this must be done independently of the specific mechanisms used to describe their behavior. In addition, if connectors were classified by this property, analyzing the correctness of a connector in an architecture would be simpler. Besides declarativity and abstraction, working with high level properties can deliver architectural mechanisms that are not bound by language features.

From the above it is clear that high level properties can have an appropriate granularity for connector description, but it is possible to go one step beyond. If these properties were thought of as primitives, connectors could be characterized by sets of property primitives, and a systematic analysis of connectors could be done.

In some manner, our proposal resembles the periodic table of elements that chemists developed a century ago. Mendeleiev constructed a table with a set of basic chemical elements in which all known substances could be decomposed. This table provided a framework for working with chemical substances, allowing new elements and substances to be discovered, and even inferred [6]. Analogously, a periodic table of connector properties would provide a framework for comparing, refining and reusing connectors. Consider two connectors characterized in terms of primitive properties, if a common subset of properties exists, a new type of connector could be factorized. Such a framework would also allow connector universe exploration, by contriving connectors as new combinations of properties.

Of course, as in Mendeleievs original table, which was, neither complete nor correct and needed refinements and additions, the periodic table of connector properties will require much work too. For instance, if one can characterize two known distinct connectors with the same set of properties, this means that the existing canonical set must be refined by adding new properties that distinguish between these connectors.

The ideas presented in this work are not completely new. In [4], a classification for architectural elements is proposed and a first sketch of the periodic table analogy is presented. The intended classification, is based on low level properties orientated towards operational description of architectural elements. This lower level approach leads the authors not to distinguish between connectors and components. We believe that our approach using high level properties, such as request/reply, synchronicity and reliability, is more adequate at an architectural level. In addition, as these properties are not applicable to components, it is reasonable to treat connectors explicitly.

In some sense the idea of decomposing connectors into properties has the same spirit as OMalley and Petersons work [5] where they present the idea of decomposing network protocols into mini and virtual protocols. Each one of these mini-protocols implements one property of the original protocol.

Another related view is given in [1], where the main goal is to understand operations over connectors. The difference resides on how this goal is achieved. [1] proposes the use of higher order connectors while we propose reusing connectors by reusing properties giving a general to do so.

In this paper we develop a framework for connector modeling following these ideas. In section 2 we present the idea of describing connectors with properties and proposed a first sketch of primitive connector properties. In addition we show how this framework can be used presenting the ideas of property refinement and connector specialization and factorization. In section 3 we discuss connector classes, and finally in section 4, we give our conclusions and point out some future work.

2 Connector Properties

A first step in the application of these ideas is to identify basic properties to characterize connectors. A good starting point is OMalley and Petersons work [5]. They present the idea of decomposing network protocols into mini and virtual protocols. Each one of these mini-protocols implements one property of the original protocol. The case study proposed in their work is based on the Remote Procedure Call protocol (RPC) and they show how this protocol can be decomposed into three distinct mini-protocols: a demultiplexing protocol that dispatches messages to the right procedure; a request/reply protocol that matches request messages with reply messages while preserving at most one semantics; and a blast algorithm with selective retransmission that fragments large messages into packets that can be transmitted on the underlying network.

This RPC decomposition is adequate from the network perspective because it follows the commonly used layered criteria of network services, but this approach is not adequate from an architectural point of view. For example decomposing large messages into packets that can be transmitted on the underlying network is not a relevant issue in the sense that components behavior is not affected by it. However, identifying connectors that provide request/reply mechanisms is crucial for architectural analysis. Other properties from the network field such as connection versus connectionless, reliable versus unreliable, and point to point versus broadcast services, are relevant from an architectural point of view.

Now, it is possible to describe a RPC connector in terms of the so far sketched candidate connector properties: RPC provides mechanisms that guarantee that for each service request exactly one service response is received; the connector provides a reliable communication in the sense that the connection is established or the source is informed of connection failure. Finally an RPC connector establishes a point to point interaction between two entities in a connectionless manner (i.e. entities do not have to manage communication sessions).

Our final goal is to establish the complete set of orthogonal connector properties allowing connector classification. We start with the following initial set of properties, that must be refined, changed and augmented. In the next sections we show some ideas of how this can be done.

Knows Target The source component must explicitly know the target component.

Request/Reply The connector guarantees that for each service request exactly one service response is received.

Synchronous For a communication to take place both source and target must be available.

Flow Control Connector allows components to stop, start and pause communication.

One Way Communication can only be done in one way.

Broadcast Connector allows multiple recipients.

Streamed Communication is based on continuos flow of data.

Connection Oriented Components must explicitly manage communication sessions.

Reliable The connector never loses information. Either the target receives transmitted data or the source is informed that the target did not receive it.

Encryption Connector has security capabilities for encryption.

Authentication Connector has security capabilities for authentication.

Compression Connector can transparently compress data for transmission.

Monitoring Connector can recognize and transmit certain classes of communication events to a monitoring component.

Typed Connector requires source and target to agree on type of information to be transmitted.

In table 1 we show how some well known connectors ([8]) can be described in terms of these properties.

Connector classification.

Property	Pipe	Remote Procedure Call (RPC)	Event Broadcast	Procedure Call (PC)	Shared Data
Knows Target	No	Yes	No	Yes	No
Request/Reply	No	Yes	No	Yes	No
Synchronous	No	Yes	No	Yes	No
Flow Control	Yes	No	No	No	No
One Way	Yes	No	No	No	No
Broadcast	No	No	Yes	No	Yes
Streamed	Yes	No	No	No	No
Connection Oriented	No	No	No	No	No
Reliable	Yes	Yes	Yes	Yes	No
Typed	No	Yes	Yes	Yes	Yes
Encryption	No	No	No	No	No
Authentication	No	No	No	No	No
Compression	No	No	No	No	No
Monitoring	No	No	No	No	No

Although the proposed table may be incomplete and the set of properties arguable, it gives us material to start working with connectors.

2.1 Property Refinement

If table 1 is inspected closely, it is possible to observe that there are two connectors that are characterized by exactly the same properties. The relationship between these connectors, RPC and PC, is very well known therefore it is reasonable that they have a common set of properties. It is also clear that these connectors are different, RPC is an inter-process interaction and PC is not, and consequently there must be properties that distinguish them. The question is which property or properties are missing from our proposed set. Will these new properties replace existing ones or should they be added to the original set. By means of these kind of questions and specific cases such as the RPC-PC, a canonical set of properties that characterizes the universe of connectors can be achieved.

2.2 Connector Specialization

If a connector is viewed as the set of properties that characterize it, we can ask ourselves what happens if we add another property. In other words what happens if we change a cell in the table from No to Yes. The result is a new connector, a specialization of the original one. For example, the Pipe connector could be refined with the Typed property, thus creating the well known Typed Pipe. Of course many specializations may not have an intuitive name, and some might not even be feasible. This last case could be pointing out the need for property refinement as explained previously.

2.3 Connector Factorization

In the same way a connector can be created by specialization, an opposite operation can be defined. Given two connectors, the common subset of properties that characterize them can be thought of as a new connector. The original connectors would be specializations of the factorized one. Finally, an interesting aspect is to analyze the relationships that specialization and factorization determine between connectors. It is obvious that a connector hierarchy is defined.

3 Connector Classes

It is clear that specializing a Pipe connector with the Compression property and with the Request/Reply property is not the same. In some sense a Compressing Pipe is still a Pipe while a *Request/Reply Pipe* is not a Pipe anymore. The fact is that there is a family of connectors that are called pipes, therefore Pipe can be considered a class of connectors rather than a connector itself. Basically, a connector class can be seen as a connector that does not define a criteria

for a set of properties, these properties are viewed as optional by the class. Consequently, a class instance is a connector with no optional properties that has all mandatory properties of its class. Table 2 represents connector classes where asterisks indicate optional properties. One can see that Pipe and Typed Pipe are instances of Class Pipe while *Request/Reply Pipe* is not.

Connector classes and instances.

Property	Pipe	RPC	Event Broadcast	PC	Shared Data	Pipe	Typed Pipe
Knows Target	No	Yes	No	Yes	No	No	No
Request/Reply	No	Yes	No	Yes	No	No	No
Synchronous	No	Yes	No	Yes	No	No	No
Flow Control	Yes	No	No	No	No	Yes	Yes
One Way	Yes	No	No	No	No	Yes	Yes
Broadcast	No	No	Yes	No	Yes	No	No
Streamed	Yes	No	No	No	No	Yes	Yes
Connection Oriented	No	No	No	No	No	No	No
Reliable	Yes	Yes	Yes	Yes	No	Yes	Yes
Typed	*	Yes	Yes	Yes	Yes	No	Yes
Encryption	*	*	*	*	*	No	No
Authentication	*	*	*	*	*	No	No
Compression	*	*	*	*	*	No	No
Monitoring	*	*	*	*	*	No	No

As for connectors, a hierarchy can be defined among classes. A specialization for a class of connectors is a subclass that has the same mandatory properties and a subset of its optional properties. In terms of table 2, a specialization of a class is obtained by changing one or more asterisks into Yes or No. In the same way, it is possible to define a factorization operation.

4 Conclusions and Future Work

In this paper we present a general framework for characterizing and operating over connectors. The spirit behind the proposed framework is similar to the one of the periodic table of chemical elements, in the sense that we wish to obtain a set of canonical properties that can describe all possible connectors and allow operations to be defined over them. We believe that this paper is an important step towards this goal.

Together with this framework we introduce the idea of describing connectors with high-level properties. We also propose a preliminary set of properties for the framework and use them to characterize some of the connectors usually found in the literature [8]. This approach led us to the notion of connector class, which we define as connectors with optional properties. In addition, we define operations

of specialization and factorization over connectors and classes and briefly discuss the relations that they determine.

We believe that the work presented in this paper opens several directions of research. First of all much is to be done on connector properties. Our proposed set of properties must be improved and in this paper we have sketched some ways of doing it. Also, as chemists did with basic elements, connector properties could be classified in some way answering questions as for example, are there incompatible sets of properties.

The set of proposed connector properties must be validated by more connector examples specially because this could lead to further analysis of the specialization and factorization relationships. We are also interested in the analysis of the relation between the operations on classes and connectors. Another idea is to use an algebraic approach, where operations could be connector constructors. With this approach one might ask, for example, if adding encryption to a typed connector is the same as typing an encrypted one.

Finally, it is very important to define a formal semantics for properties and to formalize this framework and its operations. Without a formal semantics it is very difficult to use or discuss some aspects because basic properties are too ambiguous. But formal semantics may be difficult to establish until a complete understanding of relevant properties and an interesting canonical set has been defined. This paper is a first step in this direction.

References

1. Garlan, D.: Higher Order Connectors. Workshop on Compositional Software Architecture. 1998.
2. Garlan, D., Allen, R.: Formalizing Architectural Connection. In Proceedings of the 16th International Conference on Software Engineering. 1994.
3. Garlan, D., Monroe, R., Wile, D.: ACME: An Architecture Descrip'tion Interchange Language. In Proceedings of CASCON'97. 1997.
4. Kazman, R., Clements, P., Bass, L., Abowd, G.: Classifying Architectural Elements as a Foundation for Mechanism Matching. In Proceedings of COMPSAC'97. 1997.
5. OMalley, S.W., Peterson, L.L.: A Dynamic Network Architecture. ACM Transactions on Computer Systems. 10(2):110–143. 1992.
6. Scerri, E.: The Evolution of the Periodic System. Scientific American, 279(3):56–61, September 1998.
7. Shaw, M., DeLine, R., Klein, D., Ross, T., Young, D., Zelesnik, G.: Abstractions for software architecture and tools to support them. IEEE Transactions on Software Engineering, Special Issue on Software Architecture, 21(4):314-335, April 1995.
8. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, New Jersey. 1996.