The Need to Revisit Architectural Connectors

Christos Kloukinas *

Dep. of Computing, City University, London, EC1V 0HB, U.K.

Tel: +44.20.7040.8848 Email: C.Kloukinas@soi.city.ac.uk

Abstract

Composing and analysing COTS-based critical systems depends on a great degree on the *interaction patterns* among the components and the *strategies* used by them to guarantee that the overall temporal, security and dependability requirements will be met. However, the element which naturally describes this system aspect - an architectural connector - is being continuously mistreated and underemployed. We believe that the notion of connectors needs to be revisited and we should built upon it, if we want to be able to faithfully describe complex critical systems in a manner which easily allows to identify the design problems and opportunities available to us for meeting the system requirements.

1 Introduction

Architectural connectors were quite a contentious subject in the software architecture community, with some questioning their very existence, choosing to represent them with special component types, if at all. Allen & Garlan [1] formalised the notion of a connector and identified as the most important property of checking whether a component port process p is compatible with some connector role r to be that p should *refine* r and ensure that"p must respect all of r's obligations to interact with its environment". This in short means that the port of a component (and therefore the component itself) must have a *strategy* for the *game* described by the role of the connector it is

*Partially funded by the EU project SERENITY FP6-IST-2006-27587.

assuming. Thus, a component cannot block its environment by refusing its inputs but if it considers some inputs to be undesirable then it must ensure that they become *impossible* by using an appropriate game strategy.

2 Gains from the correct use of connectors

The strategies of components can be used to represent most naturally notions such as *scheduling constraints* for temporal constraints - where the interaction described by the connector is between a system component and resource components - or for *security* and *dependability constraints* - where one needs to ensure that for each move the environment performs in the game against the system, the system can respond with one which will keep it in the set of safe states. Unfortunately, the languages currently being used and proposed for describing systems, such as UML [5] or AADL [1] [4], have only a very limited notion of connectors and fail to use them as the basis for the analysis that is needed to increase our confidence in a system. This leads to an unnecessarily complex description of systems, where the different notions and ways of controlling them are hidden and their analysis is performed in an ad-hoc manner.

We believe that in order to manage to successfully describe and manage the complex interactions among the different system components, either via explicit connectors such as RPC and pipes (or more complex protocols such as leader election, auctions, etc.) or via implicit connectors such as resource usage and other indirect channels (*tempest phenomenon*), we will need to bring the notion of connectors back into the centre stage and start treating them as first-class entities, just as important as the components themselves. Along with the reintroduction of a full connector model, our models of components will need to change so as to present in an *explicit* manner the strategies which are employed by them for the specific systems they are parts of. If we are to increase our confidence in the strategies employed by the system components, then these need to be as *modular* as possible. That is, they need to be broken down with respect to the different *system modes* and, if possible, they should be broken down with respect to each *property* that should hold for each mode, e.g., similarly to the *scheduler stack* of [3] or the controllers of [2]. We believe that such an approach would greatly help in de-obfuscating designs and in representing design decisions in a much more obvious manner than what is currently possible. In this way, designers will no longer have to entangle the strategies and the design decisions within the

[1] By the Avionics Systems Division of the Int. Soc. of Automotive Engineers.

behavioural descriptions of the system components, as they do nowadays because they have access only to simple connectors. This would make it easier to understand the *dynamics* of a system and of the strategies employed in it, make further *optimisations* easier to identify and render the whole system easier to *validate*.

3 Conclusions

The description, design and analysis of complex systems requires an easy way to describe at a high level the complicated interaction protocols that are being used in them, either explicitly or implicitly. Protocols are most naturally described by architectural connectors which currently are being either completely ignored or only supported at a very rudimentary level, making it almost impossible to describe cleanly something more complex than the usual RPC or simple message passing. By bringing connectors back into the picture, we can easily represent the particular interaction patterns and design constraints used in a system, through a more *unified* concept which will effectively be the *strategies* that the system components are employing in the protocols they are participating. By structuring these strategies according to the system *modes* and the required *properties*, we can help in making complex systems easier to *design*, *understand*, *optimise* and *validate*.

References

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. on Softw. Eng. and Methodology*, 6(3):213–249, July 1997.

[2] I. J. Hayes, M. A. Jackson, and C. B. Jones. Determining the specification of a control system from that of its environment. In *FME 2003*, volume 2805 of *LNCS*, pages 154–169, Pisa, Italy, Sept. 2003. Springer.

[3] C. Kloukinas and S. Yovine. Synthesis of safe, QoS extendible, application specific schedulers for heterogeneous real-time systems. In *ECRTS 2003*, pages 287–294, July 2003. DOI: `10.1109/EMRTS.2003.1212754` .

[4] B. A. Lewis, P. H. Feiler, and S. Vestal. The SAE architecture analysis & design language (AADL): A standard for engineering performance critical systems. In *2006 IEEE Int. Symp. on Intelligent Control*, pages 1206–1211, Oct. 2006. See also: `http://www.aadl.info`.

[5] OMG. Unified modeling language: Superstructure, version 2.1.1. Available at `http://www.omg.org/uml/` - file `formal/2007-02-05`, Feb. 2007.