Architectural Views for Computation, Coordination and Distribution - An Extended Abstract[*]

Cristóvão Oliveira[1,2] and Michel Wermelinger[1,3]

[1] Dep. de Informática, Fac. de Ciências e Tecnologia, Univ. Nova de Lisboa
2829-516 Caparica, Portugal
co@di.fct.unl.pt
[2] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
[3] Computing Department, The Open University
Walton Hall, Milton Keynes MK7 6AA, UK
m.a.wermelinger@open.ac.uk

Abstract. CommUnity and its categorical foundations provide a formal approach to Software Architecture (SA). Several concepts such as (re)configuration and (higher-order) connector have been given precise definitions in this setting. One of the cornerstones of the approach is the separation between computation, coordination and distribution. In this paper, we take this separation one step further and define explicit architectural views, one for each concern. They will be used to help to detect errors made while building the architecture. Moreover they will be a support to improve the design of the system by focusing on one concern at a time and/or by combining them with each other. In order to test the runtime behaviour of the system, the views are used as support to the distributed execution of the developed architecture.

1 Introduction

The goal of this work is to present a precise definition of three architectural views over CommUnity, a language that we have been using to support our research on the foundations of architectural description [2]. We introduce the architectural views with the aim of representing the explicit separation between computation, coordination and distribution concerns. We define transformations to obtain each view from the architectural representation we have been using so far.

Furthermore, the three views make the analysis and validation of CommUnity-based architectures easier, better structured and more incremental. For example, by focusing only on the coordination view one can easily verify indirect interactions that should not occur. Besides that, the aim behind developing

[*] This research was supported by Fundacão para a Ciência e Tecnologia through the PhD Scholarship SFRH/BD/6241/2001 of Cristóvão Oliveira

the views is to have the opportunity to concentrate only on one concern each time while modelling the architecture. Improvements on the model can be achieved studying in particular each view and this without having in mind the detail of the others.

2 Architectural Views

An architectural view describes a part of a system according to some perspective of interest, helping the designer to validate the architecture. Each view highlights and makes explicit some aspects, while omitting others.

In the setting of CommUnity [1], we are interested in three views: computation, coordination, and distribution. They should be defined in such a way as to make it easy for the designer to answer questions like:

1. which actions *cannot* occur simultaneously and which ones *must* occur together?
2. which variables are actually the same?
3. which variables are used by which actions?
4. which actions from distinct components use the same channel? are those actions synchronized?
5. which constituents (variables or actions) are always co-located?
6. which locations might have to be in touch with each other?
7. which locations (more precisely channels and actions located at those locations) are controlled by which actions?

Such checks will help the designer to validate the architecture, detect potential problems (e.g., a design with too many constituents at the same location), and take corrective actions (e.g., by relocating some of the constituents). This may require switching between views. For example, the coordination view might help the designer spot a large set of actions that always execute synchronously, whereas the distribution view shows if they are scattered among many locations or not. But this switching can be in part avoided by combining the views, meaning that, being in a given view it is possible to obtain adapted version(s) of the other(s) and combine them. Moreover the combinations appear to be useful in order to compare the views.

The combination of the computation view with the coordination view is already subsumed by the architecture, which is centered on the designs (computation view) combined with the interactions between designs (coordination view). The combination of the computation view with the distribution view doesn't add new insights either, because in the architecture each design has its channels (computation view) mapped to a location (distribution view) and each action is explicitly distributed or not over several locations. Finally, the combination of coordination with distribution is not presented because the combination of distribution and coordination is more clear and succinct.

The views are not completely independent of each other. There is some overlap between them; otherwise it would be impossible to relate the entities of one view with those of another one. In order to relate one view to another, we need unique names. Since each name is unique within each design instance in the architecture, and all views stem from the architecture, we use in views names of the form D[N].X where D is the name of the design, N is the number of the instance, and X the name of the constituent. We omit [N] if there is only one instance of design D. Moreover, the views are defined in a way as to allow their automatic construction from a given architecture.

In the graphical spirit of CommUnity, each view will be defined as a graph, with nodes aggregating actions and channels according to the perspective of interest (i.e., each node uses the usual CommUnity notation), and arcs showing relationships between such groupings.

References

1. José Luiz Fiadeiro and Tom Maibaum, *Categorical semantics of parallel program design*, Science of Computer Programming **28** (1997), 111–138.
2. José Luiz Fiadeiro, Antónia Lopes, and Michel Wermelinger, *A mathematical semantics for architectural connectors*, Generic Programming **LNCS Springer** (2003), no. 2793, 190–234.