

The CommUnity Workbench*

Cristóvão Oliveira Michel Wermelinger
Dep. de Informática, Univ. Nova de Lisboa
2829-516 Caparica, Portugal
<http://ctp.di.fct.unl.pt/~co> <http://ctp.di.fct.unl.pt/~mw>

1. Introduction

COMMUNITY proposes a formal approach to Software Architecture. However, instead of using, as Wright [1], Darwin [7] or other approaches, a process calculus to describe behaviour, it uses a parallel program design language in the style of UNITY programs [2], combining elements from IP [4]. The concepts of Software Architecture—including configuration, connection, connector, component, instantiation—are clearly defined. COMMUNITY was initially developed to show how programs fit into Goguen's categorical approach to General Systems Theory. Since then, the language and its framework have been extended to provide a formal platform for the architectural design of open and reconfigurable systems [3]. In contrast to other approaches, COMMUNITY has a richer coordination model and, even more important, it requires interaction between components to be made explicit. Besides, the COMMUNITY language takes into account the properties of the “physical” distribution topology of components and communication links. One of the main goals of COMMUNITY is to explicitly separate three architectural dimensions: Computation, Coordination, and Distribution/Mobility [5].

The COMMUNITY Workbench is being developed as a proof of concept of the COMMUNITY framework, providing an integrated graphical development environment to textually edit, debug, and run COMMUNITY programs and to graphically edit COMMUNITY software architectures.

This paper describes an extension of a previous demo we presented at ICSE'02 [9]. This extension concerns: connectors; a graphical mode to visualize and/or update interactions; an export utility to save the whole architecture or just some connectors as a textual specification which can then be easily read without the tool; the distribution and mobility constructs.

*This research was supported by Fundação para a Ciência e Tecnologia through the PhD Scholarship SFRH/BD/6241/2001 of Cristóvão Oliveira and through project POSI/32717/00 (FAST—Formal Approach to Software Architecture.), and by the European Commission through project AGILE (Architectures for Mobility), ref. IST-2001-32747 Global Computing Initiative.

In order to illustrate all the new features we adopt a simple version of a client-server system. The component client is a mobile user and the server is a printer. The communication is asynchronous and is implemented through a message passing connector using a buffer always co-located with the user. The user does not control its own location, whereas the printer, whenever it is not co-located with the user, interrupts the reception of messages and tries to move to the user's position.

2. Working with the Tool

The current version of the COMMUNITY workbench allows the designer to: specify the location dimension, then edit COMMUNITY designs, afterwards edit connectors graphically, and then define graphically the architecture, with connectors, and calculate its colimit, and finally run the colimit of the configuration as a design. The details follow.

Location Dimension: Location variables are all typed with a special sort — Location. The designer specifies the Location type and two binary relations over locations — “bt” and “reach”. The relation “bt” means that two positions are “in touch”. Coordination among components takes place only when all the locations of all the actions involved in a synchronization are “bt”. The relation “reach” means that one position is “reachable” from the other. Movement of a component to a new position is possible only when this position is reachable from the current one. The two relations are defined as boolean expressions with two parameters which are locations (see upper right corner of the figure). The designer writes the location type in the first blank text field in terms of the pre-defined types in COMMUNITY; in our example we used the type “int”. In the second and fourth field, the designer enters partial definitions of the “bt” and “reach” relations over two implicitly pre-defined location variables (“x” and “y”). In the third field, the Workbench computes the reflexive and symmetric closure of “bt”, and in the fifth field the tool adds the reflexivity condition to “reach”. Those will be the actual

expressions used to evaluate the relations.

Computation Dimension: The designer writes the designs or edits existing ones. A design consists of a set of location variables, a set of input, output or private typed variables and a set of actions. Each action has a name and a set of distributed bodies, each having a set of parallel assignments. The design is a conceptual unit of computation. COMMUNITY is independent of the actual data types used, although the workbench provides a fixed set (basic types, lists, arrays, etc.).

Coordination Dimension: In COMMUNITY, the model of interaction is based on synchronization of actions and the sharing of input/output variables between different components. The designer may select some nodes to invoke a graphic link editor to visualize and/or to set the connections between the selected nodes (lower left corner of the figure). The editor presents to the designer all the details of the selected nodes: actions, input and output variables; respectively circles, inward triangles and outward triangles. In the current version of the tool the interaction between components can be also made using connectors, which encapsulate behaviour of the interaction. The connectors are defined by a glue and a set of roles, which are designs in COMMUNITY. The insertion of a connector, in the construction of an architecture, consists in instantiating (i.e., refining) its roles with specific designs. In the demo we use two connectors: the first is an asynchronous file passing connector with three roles—sender, receiver, and buffer; the second is a distribution connector involving two components that have to be instances of the roles “chased” and “chase”, respectively. The “chase” instance is moved to the location of the instance of “chased” whenever they are not co-located. In our example the “user” refines the “chased” role and the “printer” refines the “chase” role (see the upper left corner of the figure, where the example architecture has three design instances depicted as rectangles and two connectors shown as ellipses).

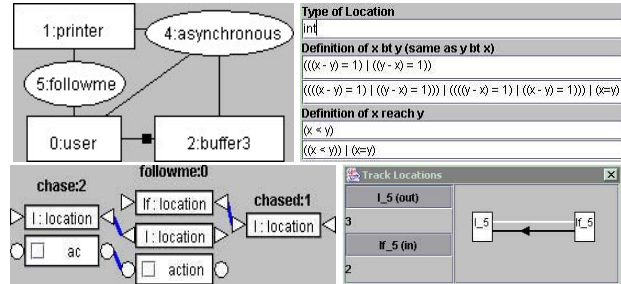
Colimit Construction: A configuration has a very precise mathematical semantics, given by a diagram in a category whose objects are the designs and whose morphisms capture a notion of program superposition [2]. Any such categorical diagram can be transformed, by a universal categorical construction called *colimit*, into a single design that represents the whole distributed system. It is important to note that the colimit is calculated by the tool; the designer does not need to have any knowledge about Category Theory to use the workbench.

Design Execution: The designer can run a design (in particular the colimit of the configuration). During the trace the designer can track the location variables. As shown in the lower right corner of the figure, the example has two location variables (“l_5” and “lf_5”) with the black line representing the “reach” relation and the white one the “bt”

relation between the values of the location variables.

We are currently planning to investigate the possibility of using a model-checking tool to verify properties of the behaviour of COMMUNITY architectures. We are also considering the implementation of high-order connectors and reconfiguration, already included in the COMMUNITY approach [6, 8].

The workbench is available from the COMMUNITY web site (www.fiadeiro.org/jose/CommUnity).



References

- [1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. on Software Eng. and Methodology*, 6(3):213-249, 1998.
- [2] K. M. Chandy and J. Misra. *Parallel Program Design—A Foundation*. Addison-Wesley, 1988.
- [3] J. L. Fiadeiro, A. Lopes and M. Wermelinger. A Mathematical Semantics for Architectural Connectors. In *Generic Programming*, LNCS 2793, pp. 190-234. Springer, 2003.
- [4] N. Francez and I. Forman. *Interacting Processes*. Addison-Wesley, 1996.
- [5] A. Lopes, J. L. Fiadeiro and M. Wermelinger. Architectural Primitives for Distribution and Mobility. In *Proc. Symp. on Foundations of Software Eng.*, pp. 41-50. ACM Press, 2002.
- [6] A. Lopes, M. Wermelinger, and J. Fiadeiro. Higher-order Connectors. *ACM Trans. on Software Eng. and Methodology*, 12(1):64-104, 2003.
- [7] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. Distributed Software Architectures In *Proc. 5th European Software Engineering Conf.*, pp. 137-153, 1995.
- [8] M. Wermelinger and J. L. Fiadeiro. A Graph Transformation Approach to Software Architecture Reconfiguration. *Science of Computer Prog.*, 44(2):133-155, August 2002.
- [9] M. Wermelinger and C. Oliveira. The CommUnity Workbench. In *Proc. 24th Intl. Conf. on Software Eng.*, p. 713. ACM Press, 2002.