Guest editors' introduction: Special issue on Process Algebra and System Architecture

M. Bernardo [a], C.A. Middelburg [b]

[a] *Univ. Urbino "Carlo Bo", Istituto S.T.I., Piazza Repubblica 13, 61029 Urbino, Italy*

[b] *Tech. Univ. Eindhoven, Comp. Sci. Dept., P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

Process algebra is a formal description technique for complex computer, communication and software systems, especially those with communicating, concurrently executing components. The key characteristics of process algebra are: (i) compositional modeling through a small number of constructs for building larger systems up from smaller ones; (ii) structural operational semantics that formally defines for each process term the labeled transition system that it stands for; and (iii) syntax-oriented and semantics-oriented behavioral reasoning via equivalences and preorders that capture the notions of same behavior and refinement, respectively.

Due to its compositional and abstract nature, process algebra naturally lends itself to being used at the architectural level of design for a complex system. At this stage, the focus is on the overall system behavior and structure, rather than on the specific features of the individual system components or the details related to the final implementation.

When working at the architectural level, based on the customer's requirements the designer should first of all identify the most appropriate architectural style according to which the system should be developed (main module-subroutines, client-server, pipe-filter, layered, object-oriented, event-based, repository, etc.). Then, following the guidelines provided by the chosen architectural style, the designer should establish the behavior of each type of component together with its interfaces with the rest of the system, as well as the way in which the instances of the component types need to be linked to each other, possibly introducing suitable connectors. This task can be carried out if an architectural description language is made available to the designer, which operates a separation of concerns between system behavior and system topology and is sufficiently easy to use. Finally, before proceeding with the implementation, the designer should check the architectural specification of the system against the customers' requirements, in a way that allows the components responsible for requirement violations to be singled out. To conduct this activity the designer needs ad hoc techniques for the detection of architectural mismatches, which can arise when assembling together several components that are correct when considered in isolation.

Building on previous work appeared in the literature – mainly architectural description languages based on process algebra and architectural analysis techniques based on equivalence checking – this special issue is devoted to recent advances in process algebra that improve its capability to deal with the architectural level of design for complex systems. Topics of interest include among others: formalization of architectural styles with process algebra; modeling support for components and connectors within process algebra; efficient techniques for architectural mismatch detection and diagnosis; methods to synthesize failure-free connectors; extensions of previous approaches to cope with dynamic and mobile architectures, support for architectural evolution, and analysis of different aspects like security, real time, performance, and dependability.

In the first paper selected for this special issue, Cuesta et al. survey most of the architectural description languages based on process algebra – Wright, PADL, LEDA, and Darwin – that have been proposed in the literature. The authors compare such languages based on the way they relate the concepts of algebraic process and architectural component, then propose a more balanced relation between the two concepts, which is called abstract process and shown through its implementation in a novel language called $\mathcal{P}i\mathcal{L}ar$. In the second paper, Bracciali et al. address the specification and verification of open systems at the architectural level of design. After showing that traditional techniques for closed systems are not adequate in this framework, the authors define a new notation called IP-calculus for the description of open systems, together with a notion of partial correctness called acceptability that can be effectively employed to verify open systems. In the third paper, Kerbœuf and Talpin consider object orientation in the context of large reactive systems, aiming at providing support for the adaptation of embedded systems with new services. This is achieved by the authors by introducing a new process calculus called Objective Signal, which mixes synchronous programming with object-oriented features – like encapsulation and behavioral inheritance – by means of suitable algebraic concurrency combinators. Finally, in the last paper Dragoni and Gaspari deal with fault tolerance aspects within distributed software architectures. To this purpose, the authors present an algebra of actors extended to model crash failures and their detection, which can be employed to assess whether certain requirements are satisfied at the architectural level even in the presence of failures.

To conclude, we would like to thank not only the authors of the above men-

tioned papers accepted for publication in this special issue, but also the contributors who submitted articles we had to leave out. We are also grateful to the following people who acted as reviewers: Roberto Barbuti, Henrik Bohnenkamp, Carlos Canal, Flavio Corradini, Rocco De Nicola, Alessandro Fantechi, Hubert Garavel, Dan Hirsch, Diego Latella, Michele Loreti, Arend Rensink, Marina Ribaudo, and Gianluigi Zavattaro. Finally, we appreciated the cooperation of Inge Bethke during the preparation of this special issue.