Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 162 (2006) 159–162
www.elsevier.com/locate/entcs

From Process Calculi to Klaim and Back

Rocco De Nicola[1,2]

Dipartimento di Sistemi e Informatica
Università di Firenze
Firenze, Italy

Abstract

We briefly describe the motivations and the background behind the design of KLAIM, a process description language that has proved to be suitable for describing a wide range of distributed applications with agents and code mobility. We argue that a drawback of KLAIM is that it is neither a programming language, nor a process calculus. We then outline the two research directions we have pursued more recently. On the one hand we have evolved KLAIM to a full-fledged language for distributed mobile programming. On the other hand we have distilled the language into a number of simple calculi that we have used to define new semantic theories and equivalences and to test the impact of new operators for network aware programming.

Keywords: Process Algebras, Network Aware Programming, Behavioural Equivalences, Formal Specifications, Systems Verification.

Introduction

In the last decade, programming computational infrastructures available globally for offering uniform services has become one of the main issues in Computer Science. The challenges come from the necessity of dealing at once with issues like communication, co-operation, mobility, resource usage, security, privacy, failures, etc., in a setting where demands and guarantees can be very different for the many different components. This has stimulated research on concepts, abstractions, models and calculi that could provide the basis for the design of systems "sound by construction", predictable and analyzable.

One of the abstractions that appears to be very important is *mobility*. This feature deeply increases flexibility and, thus, expressiveness of programming languages for network-aware programming. Evidence of the success of this programming style is provided by the recent design of commercial/prototype programming languages

[1] I would like to thank Luca Aceto and Andy Gordon for encouragement to write these notes and all organizers of the workshop for the invitation. But I need to say that I am most indebted to all researchers that through their work have contributed to the development of KLAIM and its variants.

[2] Email: denicola@dsi.unifi.it

doi:10.1016/j.entcs.2005.12.110

with primitives for moving code and processes, Java, T-Space, Oz, Pict, Oblique, Odyssey ... that have seen the involvement of several important industrial and academic research institutions.

The first foundational calculus dealing with mobility has been the π-calculus, a simple and expressive calculus aiming at capturing the essence of name passing with the minimum number of basic constructs. If considered from a network-aware perspective, one could say that π-calculus misses an explicit notion of locality and/or domain where computations take place. To overcome this deficiency of π-calculus, several foundational formalisms, presented as process calculi or strongly based on them, have been developed. We want to mention, among the others, Ambient calculus, Dπ-calculus, DJoin, Nomadic Pict, ... A major problem that has been faced in their development has been the search for the appropriate abstractions that can be considered an acceptable compromise between expressiveness, elegance, and implementability. A paradigmatic example is the Ambient calculus: it is very elegant and expressive, but a reasonable distributed implementation is still problematic.

A Kernel Language for Agents Interaction and Mobility

KLAIM (Kernel Language for Agents Interaction and Mobility) is a formalism specifically designed to describe distributed systems made up of several mobile interacting components that is positioned along the same lines of the above mentioned calculi. The distinguishing features of the approach is the explicit use of localities for accessing data or computational resources. The choice of its primitives was heavily influenced by *CCS* and π-calculus and by Linda. Indeed, KLAIM stemmed from our work on process algebras with localities [4] and our work on the formalization of the semantics of Linda as a process algebra [10].

Linda is a coordination language that relies on an asynchronous and associative communication mechanism based on a shared global environment called *tuple space*, a multiset of *tuple*s. Tuples are ordered sequence of information items (called *fields*). There can be *actual fields* (i.e., expressions, processes, localities, constants, identifiers) and *formal fields* (i.e., variables). Tuples are *anonymous* and *content-addressable*. The basic interaction mechanism is *pattern–matching* that is used to select tuples from tuple spaces. Linda has four primitives for manipulating tuple spaces: two blocking operations that are used for accessing and removing tuples and two non-blocking ones that are used for adding tuples.

KLAIM can be seen as an asynchronous higher–order process calculus whose basic actions are the original Linda primitives enriched with explicit information about the location of the nodes where processes and tuples are allocated. Communications take place through distributed repositories and remote operations. The primitives allow programmers to distribute and retrieve data and processes to and from the different localities (nodes) of a net. Localities are first-class citizens that can be dynamically created and communicated. Tuples can contain both values and code that can be subsequently accessed and evaluated. An allocation environment, associating logical and physical localities, is used to free the programmer from

considering the precise physical allocation of the distributed tuple spaces.

The main drawback of KLAIM is that it is neither an actual programming language nor a process calculus. We have thus, more recently, worked along two directions. On the one hand, we have evolved KLAIM to a full-fledged language (X-KLAIM) to be used for distributed mobile programming. On the other hand, we have distilled the language into a number of simpler calculi that we have used to define new semantic theories and equivalences and to assess the expressive power of tuple based communications and evaluate the theoretical impact of new linguistic primitives.

A Programming Language based on Klaim

X-KLAIM [1] is an experimental programming language that has bee specifically designed to program distributed systems with several components interacting through multiple tuple spaces and mobile code (possibly object-oriented). X-KLAIM has been implemented on the top of a run-time system that was developed in Java for the sake of portability [2]. The linguistic constructs of KLAIM have proved to be appropriate for programming a wide range of distributed applications with agents and code mobility that, once compiled in Java, can run over different platforms.

Klaim-based Calculi

From KLAIM we have distilled μKLAIM, cKLAIM and lcKLAIMand we have studied the encoding of each of them into a simpler one [7]. μKLAIM is obtained from KLAIM by eliminating from the distinction between logical and physical localities (i.e., *no allocation environment*) and the possibility of higher order communication (*i.e., no process code in tuples*). cKLAIM, is obtained from μKLAIM by only considering *monadic* communications and by removing the `read` action, the non destructive variant of the `in` basic actions. lcKLAIM is obtained from cKLAIM by removing also the possibility of performing remote inputs and outputs; communications is only local and process migration is needed to use remote resources.

This work on core calculi has also stimulated and simplified the search for other variants of KLAIM that better model more sophisticated settings for network aware programming. In [6] and in [8] we have considered TOPOLOGICAL-KLAIM a variant of cKLAIM that permits explicit creation of inter-node connections and their destruction and thus considering two typical features of global computers, namely *dynamic inter-node connections* and *failures*. In [9] we have developed more flexible (but still easily implementable) forms of pattern matching.

For the simpler calculi we have been able to apply the theory developed in [3] and to introduce two abstract semantics, *barbed congruence* and *may testing*. They are obtained as the closure under operational reductions and/or language contexts of the extensional equivalences induced by what can be considered a *basic observation* for global computers that aims at testing whether

`a specific site is up and running`

and amounts to testing whether a site provides a data of any kind.

For the two equivalences obtained as context closures, we have also provided alternative characterizations that permit a better appreciation of their discriminating power and the development of proof techniques that avoid universal quantification over contexts. Indeed, we have established their correspondence with a bisimulation-based and a trace-based equivalence over the labelled transition system used to describe the semantics for the different variants of KLAIM.

Information, software and papers related to KLAIM and the KLAIM Project can be retrieved at: http://music.dsi.unifi.it/klaim.html.

References

[1] L. Bettini, R. De Nicola. Interactive Mobile Agents in X-KLAIM. In *SFM-05:Moby, 5th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Mobile Computing*, volume 3465 of *LNCS*, pages 29–68, Spinger, 2005 .

[2] L. Bettini, R. De Nicola, and R. Pugliese. KLAVA: a Java Package for Distributed and Mobile Applications. *Software – Practice and Experience*, 32:1365–1394, 2002.

[3] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Basic observables for processes. *Inf. Comput.*, 149(1):77–98, 1999.

[4] Flavio Corradini and Rocco De Nicola. Locality based semantics for process algebras. *Acta Inf.*, 34(4):291–324, 1997.

[5] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Trans. on Software Engineering*, 24(5):315–330, 1998.

[6] R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. In C. Palamidessi et al., editor, *Proc. ICALP 2005*, volume 3580 of LNCS, page 1226-1238 Springer, 2005.

[7] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of KLAIM-based calculi. To appera in TCS. Short version in *Proc. of EXPRESS'04*, ENTCS 128(2):117–130. Elsevier, 2004.

[8] R. De Nicola, D. Gorla, and R. Pugliese. Global computing in a dynamic network of tuple spaces. In *Proc. of COORDINATION'05*, volume 3454 of *LNCS*, pages 157–172. Springer, 2005.

[9] R. De Nicola, D. Gorla, and R. Pugliese. Pattern matching over a dynamic network of tuple spaces. In *Proc. of FMOODS'05*, volume 3535 of *LNCS*, pages 1–14. Springer.

[10] Rocco De Nicola and Rosario Pugliese. Linda-based applicative and imperative process algebras. *Theor. Comput. Sci.*, 238(1-2):389–437, 2000.