Science of Computer Programming 117 (2016) 17–44

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A type-sound calculus of computational fields ☆,☆☆

Ferruccio Damiani [a,*], Mirko Viroli [b], Jacob Beal [c]

[a] University of Torino, Italy
[b] University of Bologna, Italy
[c] Raytheon BBN Technologies, USA

ARTICLE INFO

Article history:
Received 14 February 2014
Received in revised form 31 October 2015
Accepted 9 November 2015
Available online 2 December 2015

Keywords:
Computational field
Core calculus
Operational semantics
Spatial computing
Type soundness

ABSTRACT

A number of recent works have investigated the notion of "computational fields" as a means of coordinating systems in distributed, dense and dynamic environments such as pervasive computing, sensor networks, and robot swarms. We introduce a minimal core calculus meant to capture the key ingredients of languages that make use of computational fields: functional composition of fields, functions over fields, evolution of fields over time, construction of fields of values from neighbours, and restriction of a field computation to a sub-region of the network. We formalise a notion of type soundness for the calculus that encompasses the concept of domain alignment, and present a sound static type inference system. This calculus and its type inference system can act as a core for actual implementation of coordination languages and models, as well as to pave the way towards formal analysis of properties concerning expressiveness, self-stabilisation, topology independence, and relationships with the continuous space–time semantics of spatial computations.

1. Introduction

In a world ever more densely saturated with computing devices, it is increasingly important to have effective tools for developing coordination strategies that can govern collections of these devices [8]. The goals of such systems are typically best expressed in terms of operations and behaviours over aggregates of devices, e.g., "send a tornado warning to all phones in the forecast area," or "activate all displays guiding me along a route towards the nearest group of my friends." The available models and programming languages for constructing distributed systems, however, have generally operated at the level of individual devices and their interactions, thereby obfuscating the design process. Effective models and programming languages are needed to allow the construction of distributed systems at the natural level of aggregates of devices. These

☆ This work has been partially supported by project HyVar (www.hyvar-project.eu, this project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644298—Damiani), by EU FP7 project SAPERE (www.sapere-project.eu, under contract No. 256873—Viroli), by ICT COST Action IC1402 ARVI (www.cost-arvi.eu—Damiani), by ICT COST Action IC1201 BETTY (www.behavioural-types.eu—Damiani), by the Italian MIUR PRIN 2010/2011 2010LHT4KM project CINA (sysma.imtlucca.it/cina—Damiani & Viroli), by Ateneo/CSP project RunVar (Damiani), and by the United States Air Force and the Defense Advanced Research Projects Agency under Contract No. FA8750-10-C-0242 (Beal). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views, opinions, and/or findings contained in this article are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for public release; distribution is unlimited.

☆☆ This paper was submitted for the special issue, "Foundations of Coordination Languages and Software (FOCLASA 2013)".

* Corresponding author.
E-mail addresses: ferruccio.damiani@unito.it (F. Damiani), mirko.viroli@unibo.it (M. Viroli), jakebeal@bbn.com (J. Beal).

http://dx.doi.org/10.1016/j.scico.2015.11.005
0167-6423/

must also be associated with a global-to-local mapping that links the aggregate-level specification to the operations and interactions of individual devices that are necessary to implement it.

Recently, approaches based on models of computation over continuous space and time have been introduced, which promise to deliver aggregate programming capabilities for the broad class of *spatial computers* [10]: networks of devices embedded in space, such that the difficulty of moving information between devices is strongly correlated with the physical distance between devices. Examples of spatial computers include sensor networks, robot swarms, mobile ad-hoc networks, reconfigurable computing, emerging pervasive computing scenarios, and colonies of engineered biological cells.

A large number of formal models, programming languages, and infrastructures have been created with the aim of supporting computation over space–time, surveyed in [7]. Several of these are directly related to the field of coordination models and languages, such as the pioneer model of TOTA [29], the (bio)chemical tuple-space model [47], the $\sigma\tau$-Linda model [50], and the pervasive ecosystems model in [52]. Their recurrent core idea is that through a process of diffusion, recombination, and composition, information injected in one device (or a few devices) can produce global, dynamically evolving *computational fields*—functions mapping each device to a structured value. Such fields are aggregate-level distributed data structures which, due to the ongoing feedback loops that produce and maintain them, are generally robust to changes in the underlying topology (e.g., due to faults, mobility, or openness) and to unexpected interactions with the external environment. They are thus useful for implementing and composing self-organising coordination patterns to adaptively regulate the behaviour of complex distributed systems [29,47,48].

A sound engineering methodology for space–time coordination systems will require more than just specification, but also the ability to predict to a good extent the behaviour of computational fields from the underlying local interaction rules—a problem currently solved only for a few particular cases (e.g., [6,3]). This paper contributes to that goal by:

1. Introducing the *computation field calculus (*CFC*)*, a minimal core calculus meant to precisely capture a set of key ingredients of programming languages supporting the creation of computational fields: composition of fields, functions over fields, evolution of fields over time, construction of fields of values from neighbours, and restriction of a field computation to a sub-region of the network.
2. Formalising a notion of type soundness for CFC that encompasses the concept of *domain alignment* (i.e., proper sharing of information between devices), and presenting a sound static type inference system supporting polymorphism à la ML [17]. The main challenges in the design of the type system are to ensure domain alignment (which is complicated by the fact that the same expression may be evaluated many times, even recursively) and to support polymorphism à la ML without breaking domain alignment.

CFC is largely inspired by Proto [5,33], the archetypal spatial computing language (and is in fact a much simpler fragment of it). As with Proto, it is based on the idea of expressing aggregate system behaviour by a functional composition of operators that manipulate (evolve, combine, restrict) continuous fields. Critically, these specifications can be also interpreted as local rules on individual devices, which are iteratively executed in asynchronous "computation rounds", comprising reception of all messages from neighbours, computing the local value of fields, and spreading messages to neighbours. The operational semantics of the proposed calculus precisely models single device computation, which is ultimately responsible for all execution in the network. The distinguished interaction model of this approach, which is formalised into a calculus in this paper, is based on representing state and message content in an unified way as an annotated evaluation tree. Field construction, propagation, and restriction are then supported by local evaluation "against" the evaluation trees received from neighbours. Not only is field calculus much simpler than Proto (and thus a tractable target for analysis), but the proposed formalisation also goes beyond Proto (which is a dynamically typed language) by introducing a static type inference system and a type soundness property that encompasses the notion of domain alignment, thereby enabling static analysis of the soundness and resilience properties of field computations.

The work thus developed formalises key constructs of existing coordination languages or models targeting spatial computing. As such, we believe that the calculus and its type inference system pave the way towards formal analysis of key properties applicable to various coordination systems, concerning expressiveness, self-stabilisation, topology independence, and relationships with the continuous space–time semantics of spatial computations.

The remainder of the paper is organised as follows: Section 2 describes the linguistic constructs of CFC and their application to system coordination. Section 3 illustrates how single devices interpret the CFC constructs locally. Section 4 illustrates some examples of programs to show the expressiveness of CFC. Section 5 formalises the operational semantics of CFC. Section 6 illustrates some examples of ill-formed programs to motivate the design of the type system. Section 7 presents the type inference system and its key properties (domain alignment and type soundness). Section 8 briefly surveys the main elements of a toolchain that is under development and grounds on CFC. Finally, Section 9 discusses related work and Section 10 concludes by outlining possible directions for future work. The appendix contains the proofs of the main results.