









# On a Higher-Order Calculus of Computational Fields

Giorgio Audrito<sup>1</sup> , Mirko Viroli<sup>2</sup> , Ferruccio Damiani<sup>1</sup>  ,  
Danilo Pianini<sup>2</sup> , and Jacob Beal<sup>3</sup> 

<sup>1</sup> Dipartimento di Informatica, University of Turin, Turin, Italy  
{giorgio.audrito,ferruccio.damiani}@unito.it

<sup>2</sup> DISI, University of Bologna, Cesena, Italy  
{mirko.viroli,danilo.pianini}@unibo.it

<sup>3</sup> Raytheon BBN Technologies, Cambridge, MA, USA  
jakebeal@ieee.org

**Abstract.** Computational fields have been proposed as an effective abstraction to fill the gap between the macro-level of distributed systems (specifying a system’s collective behaviour) and the micro-level (individual devices’ actions of computation and interaction to implement that collective specification), thereby providing a basis to better facilitate the engineering of collective APIs and complex systems at higher levels of abstraction. This approach is particularly suited to complex large-scale distributed systems, like the Internet-of-Things and Cyber-Physical Systems, where new mechanisms are needed to address composability and reusability of collective adaptive behaviour. This work introduces a full formal foundation for field computations, in terms of a core calculus equipped with typing, denotational, and operational semantics. Critically, we apply techniques for formal programming languages to collective adaptive systems: we provide formal establishment of a link between the micro- and macro-levels of collective adaptive systems, via a result of computational adequacy and abstraction for the (aggregate) denotational semantics with respect to the (per-device) operational semantics.

**Keywords:** Distributed computing · Core calculus · Type system · Denotational semantics · Operational semantics · Computational adequacy

## 1 Background

Aggregate computing [6] is a paradigm aiming to address the complexity of large-scale distributed systems, by means of the notion of *computational field* [15] (or simply *field*): this is a collective, distributed map from computational events (when and where a device executes a computational action, also called a *round*) to computational objects (data values of any sort, including higher-order objects such as functions and processes) representing the result of computation at that

event. Computing with fields means computing such global data structures, and defining a reusable block of behaviour means to define a reusable computation from fields to fields. This functional view holds at any level of abstraction, from low-level mechanisms of the language up to whole applications, which ultimately work by getting input fields from sensors and processing them to produce output fields for actuators. Most importantly, computing with fields is functional and hence declarative: (i) the designer focusses on the intended global goal of system behaviour, while the dynamics of interactions is left to the underlying platform (i.e., semantics); and (ii) one can scale with complexity by relying on functional composition: libraries of reusable building blocks can be constructed, and successive layering can be used to bottom-up derive whole applications.

The *field calculus* [11] is a tiny functional language providing basic constructs to work with fields.<sup>1</sup> It provides a unifying approach to understanding and analysing the wide range of approaches to distributed systems engineering that make use of computational fields [5, 21]. The operational semantics of field calculus [11] can act as a blueprint for actual implementations where myriad devices interact via proximity-based broadcasts. More recently, the field calculus has been used to investigate formal properties of resiliency to environment changes [18, 20] and to device distribution [7]. Its expressiveness has been investigated by introducing the *cone Turing Machine* [1].

The *higher-order field calculus* [12] combines self-organisation and code mobility by extending the field calculus with a semantics for distributed first-class functions. It allows self-organisation code to be naturally handled like any other data, e.g., dynamically constructed, compared, spread across devices, and executed in safely encapsulated distributed scopes. Ultimately, this calculus provides programmers with a novel first class abstraction, a “distributed function field”. This is a dynamically evolving map from a network of devices to a set of executing distributed processes: in each space-time region where the process is the same, devices form a coalition collectively carrying on that process in isolation.

## 2 Contributions of [3]

This paper presents syntax and operational semantics of the higher-order field calculus together with new contributions: a type system for the higher-order version of the calculus, a denotational semantics, and associated properties. The new, enhanced syntax is parametric in the set of the modeled data values (in [12] Booleans, numbers, and pairs were explicitly modeled). Moreover, the `if` construct has been removed by encapsulating its branching capability into function calls, which now take the form of a function field applied to arguments, implicitly enacting branching. Then, a novel key insight and technical result of this paper is that the notoriously difficult problem of reconciling local and global behaviour in a complex adaptive system [20] can be connected to a well-known problem in programming languages: correspondence between denotational and

<sup>1</sup> Much as  $\lambda$ -calculus [9] captures the essence of functional computation and FJ [14] the essence of class-based object-oriented programming.

operational semantics. On the one hand, denotational semantics can be used to characterise computations in terms of their collective effect across space (available devices) and time (device computation events)—i.e., the macro level. On the other hand, operational semantics gives a transition system dictating each device’s individual and local computing/interactive behaviour—i.e., the micro level. Correspondence between the two, formally proved in this paper via *computational adequacy* and a form of *abstraction* (c.f. [10, 19]) that we call *computational abstraction*, thus provides a formal micro–macro connection: one designs a system considering the denotational semantics of programming constructs, and an underlying platform running the distributed interpreter defined by the operational semantics guarantees a consistent execution. This is a significant step towards effective methods for the engineering of self-adaptive systems, achieved thanks to the standard theory and framework of programming languages.

### 3 Conclusions, Related and Future Work

The work presented in this paper builds on a sizable body of prior work, for which the field calculus can somewhat act as a lingua franca: foundational approaches to group interaction (ambients [8], shared-spaces [22]), device abstraction languages (TOTA [15], Hood [23]), pattern languages [16], information movement languages [17], and spatial computing languages (MGS [13] and Proto [4]). Accordingly, future plans include consolidation of this work to investigate variants of the field calculus [2], to support an analytical methodology and a practical toolchain for system development, and to isolate fragments of the calculus that satisfy behavioural properties such as self-stabilisation developed in [20].

### References

1. Audrito, G., Beal, J., Damiani, F., Viroli, M.: Space-time universality of field calculus. In: Di Marzo Serugendo, G., Loreti, M. (eds.) COORDINATION 2018. LNCS, vol. 10852, pp. 1–20. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-92408-3\\_1](https://doi.org/10.1007/978-3-319-92408-3_1)
2. Audrito, G., Damiani, F., Viroli, M., Casadei, R.: Run-time management of computation domains in field calculus. In: 2016 IEEE 1st International Workshops on Foundations and Applications of Self\* Systems (FAS\*W), pp. 192–197. IEEE (2016). <https://doi.org/10.1109/FAS-W.2016.50>
3. Audrito, G., Viroli, M., Damiani, F., Pianini, D., Beal, J.: A higher-order calculus of computational fields. *ACM Trans. Comput. Logic* **20**(1), 5:1–5:55 (2019). <https://doi.org/10.1145/3285956>
4. Bachrach, J., Beal, J., McLurkin, J.: Composable continuous space programs for robotic swarms. *Neural Comput. Appl.* **19**(6), 825–847 (2010)
5. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: languages for spatial computing. In: Mernik, M. (ed.) *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, chap. 16, pp. 436–501. IGI Global (2013). <https://doi.org/10.4018/978-1-4666-2092-6.ch016>
6. Beal, J., Pianini, D., Viroli, M.: Aggregate programming for the internet of things. *IEEE Comput.* **48**(9) (2015). <https://doi.org/10.1109/MC.2015.261>

7. Beal, J., Viroli, M., Pianini, D., Damiani, F.: Self-adaptation to device distribution in the internet of things. *ACM Trans. Auton. Adapt. Syst.* **12**(3), 12:1–12:29 (2017). <https://doi.org/10.1145/3105758>
8. Cardelli, L., Gordon, A.D.: Mobile ambients. *Theor. Comput. Sci.* **240**(1), 177–213 (2000). [https://doi.org/10.1016/S0304-3975\(99\)00231-5](https://doi.org/10.1016/S0304-3975(99)00231-5)
9. Church, A.: A set of postulates for the foundation of logic. *Ann. Math.* **33**(2), 346–366 (1932). <https://doi.org/10.2307/1968337>
10. Curien, P.: Definability and full abstraction. *Electronic Notes Theoretical Comput. Sci.* **172**, 301–310 (2007). <https://doi.org/10.1016/j.entcs.2007.02.011>
11. Damiani, F., Viroli, M., Beal, J.: A type-sound calculus of computational fields. *Sci. Comput. Program.* **117**, 17–44 (2016). <https://doi.org/10.1016/j.scico.2015.11.005>
12. Damiani, F., Viroli, M., Pianini, D., Beal, J.: Code mobility meets self-organisation: a higher-order calculus of computational fields. In: Graf, S., Viswanathan, M. (eds.) *FORTE 2015*. LNCS, vol. 9039, pp. 113–128. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-19195-9\\_8](https://doi.org/10.1007/978-3-319-19195-9_8)
13. Giavitto, J.L., Godin, C., Michel, O., Prusinkiewicz, P.: Computational models for integrative and developmental biology. Technical report, 72-2002, Univerite d'Evry, LaMI (2002)
14. Igarashi, A., Pierce, B.C., Wadler, P.: Featherweight Java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.* **23**(3), 396–450 (2001). <https://doi.org/10.1145/503502.503505>
15. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications: the TOTA approach. *ACM Trans. Softw. Eng. Methodol.* **18**(4), 1–56 (2009). <https://doi.org/10.1145/1538942.1538945>
16. Nagpal, R.: Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics. Ph.D. thesis, MIT, Cambridge, MA, USA (2001)
17. Newton, R., Welsh, M.: Region streams: functional macroprogramming for sensor networks. In: *Workshop on Data Management for Sensor Networks*, pp. 78–87. ACM, August 2004. <https://doi.org/10.1145/1052199.1052213>
18. Nishiwaki, Y.: Digamma-calculus: a universal programming language of self-stabilizing computational fields. In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self\* Systems (FAS\*W)*. IEEE (2016). <https://doi.org/10.1109/FAS-W.2016.51>
19. Stoughton, A.: Fully abstract models of programming languages. *Research Notes in Theoretical Computer Science*, Pitman (1988)
20. Viroli, M., Audrito, G., Beal, J., Damiani, F., Pianini, D.: Engineering resilient collective adaptive systems by self-stabilisation. *ACM Trans. Model. Comput. Simul.* **28**(2), 16:1–16:28 (2018). <https://doi.org/10.1145/3177774>
21. Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From field-based coordination to aggregate computing. In: Di Marzo Serugendo, G., Loreti, M. (eds.) *COORDINATION 2018*. LNCS, vol. 10852, pp. 252–279. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-92408-3\\_12](https://doi.org/10.1007/978-3-319-92408-3_12)
22. Viroli, M., Casadei, M., Montagna, S., Zambonelli, F.: Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Trans. Auton. Adapt. Syst.* **6**(2), 14:1–14:24 (2011). <https://doi.org/10.1145/1968513.1968517>
23. Whitehouse, K., Sharp, C., Brewer, E., Culler, D.: Hood: a neighborhood abstraction for sensor networks. In: *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*. ACM Press (2004). <https://doi.org/10.1145/990064.990079>