Predictable Self-Organization with Computational Fields

Jacob Beal
Raytheon BBN Technologies
Cambridge, MA, USA
Email: jakebeal@bbn.com

Mirko Viroli
University of Bologna
Bologna, Italy
Email: mirko.viroli@unibo.it

I. Aggregate Programming

In recent years, a number of different strands of research on self-organizing systems have come together to create a new "aggregate programming" approach to the engineering of distributed systems. Aggregate programming is motivated by a desire to avoid the notoriously intractable "local to global" problem, where the system designer must predict how to control individual devices to achieve a collective goal. Instead, the designer programs an abstraction of the collective, composing "building block" primitives from a library of special cases where the local-to-global problem is already solved.

Unifying a number of the proposed aggregate programming approaches is the notion of a "computational field" that maps each device in the field's domain to a local value in its range. This concept was originally developed for spatial computers, in which communication and geometric position are closely linked, but can support effective aggregate programming of many non-spatial networks as well. A mathematical foundation for such approaches has been formalized recently with a minimal "field calculus" that appears to be an effective unifying model, covering a wide range of aggregate programming models, both continuous (e.g., geometry-based) and discrete (e.g., graph-based).

On this foundation, restricted languages can ensure various desirable properties such as scalability, self-stabilization, and robustness to perturbation. By building up a sufficiently broad collection of composable "building block" distributed algorithms, it is possible to enable simple and rapid development of complex distributed systems that are implicitly scalable and resilient. The ultimate aim of this line of research is to make the programming of robust distributed systems as simple and widespread as single-processor programming, thereby enabling widespread increases in the reliability, efficiency, and democracy of our technological infrastructure.

II. Background and Application Domains

Aggregate programming and the computational field model emerge from the commonalities of a number of different efforts across many application domains, particularly those where there is a strong link between communication and the spatial distribution of devices. In a number of fields, including sensor networks, high-performance computing, biological engineering and modeling, swarm robotics, and pervasive systems, there has been a recognition that programming efficient and resilient systems at the level of individual devices and their interactions is typically extremely difficult, error prone, and costly for analysis and proofs. In all of these domains, researchers have investigated diverse approaches to raising the abstraction level of programming by describing the behaviors of aggregates.

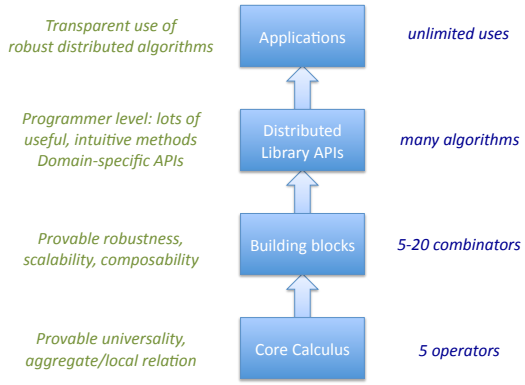

Fig. 1. The general program of aggregate programming research: building on a minimal computational foundation that connects local and aggregate behavior, it is possible to engineer collections of composable "building block" algorithms ensuring all constructions of such building blocks have desirable scalability and resilience properties. Libraries built from such building blocks can then provide a simple interface for constructing a broad range of implicitly scalable and resilient distributed systems.

Recently, however, it has begun to be recognized that there are commonalities that tend to be shared across all of these domains. A detailed survey and analysis of aggregate programming approaches across a number of fields, in [1], identified several such common elements. Field calculus aims to capture the enabling commonalities that lie at the foundation of many of these approaches.

III. Field Calculus and Continuum Models

The computational field calculus, introduced in [2], is a minimal model for aggregate programming. It is based on the notion of a computational field, generalized from the physics notions of scalar and vector fields: a computational field is a function that maps from a set of devices to some data value held at each device.

Field calculus is based on five constructs, all of which can be equivalently treated as either aggregate operations or unsynchronized actions by individual devices in the aggregate. Two are ordinary programming constructs: "built-in" operators provide stateless local functions, such as addition, comparison, and reading sensors, while function definition and call support abstraction and recursion. The other three constructs manage aggregate-specific operations: `nbr` shares information between devices, `if` restricts the domain of computations, and `rep` maintains and updates distributed state information. A practical implementation of field calculus is available in the Proto programming language [3] and the virtual machine, libraries, and simulator accompanying its implementations.

Importantly, field calculus is also space-time universal, meaning that it can approximate any causal and approximable space-time computation, whether discrete or continuous [4]. This is significant because many aggregate computations are closely linked to physical notions of space and time. For example, sensor networks are typically distributed to sample properties of a continuous environment, and swarms of UAVs maneuver through a continuous space. Continuum models such as the amorphous medium abstraction [5] take advantage of this, viewing a network of computational devices as a discrete approximation of a space-filling computational material. This is useful both for carrying out aggregate operations formulated as geometric constructions, and for simplifying analysis of many distributed algorithms, which helps to enable the development of mechanisms for construction of resilient and scalable systems.

IV. Systems of Robust "Building Blocks"

Field calculus provides a solid foundation linking aggregate and local views, but its universality makes it in some ways too general: field calculus provides the ability to express any aggregate program, even fragile or badly behaving programs. If distributed programming is to be simple, then it needs to be easy to express safe and resilient programs, and to analyze programs to determine whether they are safe and resilient.

Towards this end, taxonomies and syntaxes of "building block" distributed algorithms are being constructed. The first [6] aimed simply to identify key biologically-inspired self-organization techniques and to create a taxonomy explaining their relationships. More powerful, however, are syntaxes of building blocks, where it is possible to prove that every program expressed in the given syntax has certain desirable properties. For example, the simple spreading and aggregation syntax in [7] provides the guarantee that any program in that syntax will be self-stabilizing, meaning that it will converge to a correct set of states within finite time. In [8], a set of five higher-level building blocks provide broader capabilities while ensuring self-stabilization in time proportional to the network diameter. Stronger properties can be obtained as well, such as scalability and resilience against changes to the position or identity of devices comprising the network.

These building block syntaxes are necessarily terse and use somewhat obscure and highly generalized elements, in order to be compact enough for mathematical proofs to be tractable. They are thus generally best used not directly, but rather specialized and composed together into APIs with more intuitive functions, which also provide the desired properties, by virtue of being expressible in the building block syntax. For example, in [8] one of the five building-block operators is `G`, a highly generalized operator that computes gradient-ascending path integrals. By setting its parameters, however, this can be specialized into functions such as distributed distance measurement, information broadcast, and path-directed forecasting. It is APIs of such building-block-derived algorithms, then, that are intended to be the interface that allows non-specialist programmers to readily construct distributed systems that are implicitly scalable and self-stabilizing.

V. Future Directions

Accessible libraries built on top of building-block syntaxes are currently embryonic at best. One of the key future directions for this area is the elaboration of such libraries, both generic and domain-specific. Work to date has also treated primarily static code on static devices: future work will extend results to handle higher-order and first-class functions, mobile code, and mobile devices such as UAVs and smart-phones. It will also be important to continue extending both the coverage of the building block syntaxes and the properties that can be guaranteed for such syntaxes. Finally, as the capabilities of aggregate programming are becoming both useful and usable for end-domain applications, it can start to be applied practically to improve a wide range of complex engineered systems.

References

[1] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, "Organizing the aggregate: Languages for spatial computing," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, M. Mernik, Ed. IGI Global, 2013, ch. 16, pp. 436–501, a longer version available at: `http://arxiv.org/abs/1202.5509`.

[2] M. Viroli, F. Damiani, and J. Beal, "A calculus of computational fields," in *Advances in Service-Oriented and Cloud Computing*, ser. Communications in Computer and Information Sci., C. Canal and M. Villari, Eds. Springer Berlin Heidelberg, 2013, vol. 393, pp. 114–128.

[3] J. Beal and J. Bachrach, "Infrastructure for engineered emergence in sensor/actuator networks," *IEEE Intelligent Systems*, vol. 21, pp. 10–19, March/April 2006.

[4] J. Beal, M. Viroli, and F. Damiani, "Towards a unified model of spatial computing," in *7th Spatial Computing Workshop (SCW 2014)*, AAMAS 2014, Paris, France, May 2014.

[5] J. Beal, "Programming an amorphous computational medium," in *Unconventional Programming Paradigms*, ser. Lecture Notes in Computer Science, J.-P. Banatre, P. Fradet, J.-L. Giavitto, and O. Michel, Eds. Springer Berlin Heidelberg, 2005, vol. 3566, pp. 121–136. [Online]. Available: http://dx.doi.org/10.1007/11527800_10

[6] J. Fernandez-Marquez, G. di Marzo Serugendo, S. Montagna, M. Viroli, and J. Arcos, "Description and composition of bio-inspired design patterns: a complete overview," *Natural Computing*, vol. 12, no. 1, pp. 43–67, 2013.

[7] M. Viroli and F. Damiani, "A calculus of self-stabilising computational fields," in *Coordination Languages and Models*, ser. LNCS, eva Kühn and R. Pugliese, Eds. Springer-Verlag, Jun. 2014, vol. 8459, pp. 163–178, proceedings of the 16th Conference on Coordination Models and Languages (Coordination 2014), Berlin (Germany), 3-5 June.

[8] J. Beal and M. Viroli, "Building blocks for aggregate programming of self-organising applications," in *2nd FoCAS Workshop on Fundamentals of Collective Systems*, SASO 2014, London, UK, September 2014.