2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)

Practical aggregate programming with Protelis

Danilo Pianini
University of Bologna
Cesena, Italy
danilo.pianini@unibo.it

Jacob Beal
Raytheon BBN Technologies
Cambridge, MA, US
jakebeal@ieee.org

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

***Abstract*—Collective adaptive systems are an emerging class of networked and situated computational systems with a wide range of applications, such as in the Internet of Things, wireless sensor networks, and smart cities. Engineering such systems poses a number of challenges, and in particular many approaches, based upon designing the machine-to-machine interaction directly, suffer from a local-to-global abstraction problem. In this tutorial, we introduce the aggregate computing approach, rooted in the field calculus and practically available through the Protelis programming language, as a means to build collective, situated adaptive systems. The approach focuses on programming the overall aggregate behaviour, making use of a "resilience API," while leaving to these libraries and the language machinery the responsibility of mapping this to the behavior of individual devices.**

I. Introduction

Aggregate computing is a novel programming paradigm whose foundational concept is the so-called computational field, a data structure situated in space and time, which maps from a set of devices to data values held at each. This approach is particularly valuable for scenarios where distribution and situatedness are key, such as those common in the Internet of Things [1], wireless sensor networks [2], and smart cities [3]. Its foundation is field calculus [4] and a system of "building block" operators providing resilient properties such as self-stabilization [5], and independence from the device distribution changes [6]. Protelis provides a practical implementation of this language [7], implementing the field calculus with a dynamic, Java Virtual Machine-hosted language featuring Java interoperability and a rich library of functions [8].

II. Computational model

In the field calculus (and derived languages) each device asynchronously executes computational rounds composed of three phases: *(i)* collect shared-state messages received from other devices; *(i)* compute its own program; *(i)* send its own shared state to neighbors. A valid program is composed of the basic mechanisms of function and value definition (being higher order, functions are values too), built-in operators (such as addition, subtraction, method invocation), and of three operators specific to field calculus:

- **`rep`** maintains and updates distributed state information, allowing for stateful computation;
- **`nbr`** shares and collects information across neighboring devices;
- **`if`** restricts the domain of a computation.

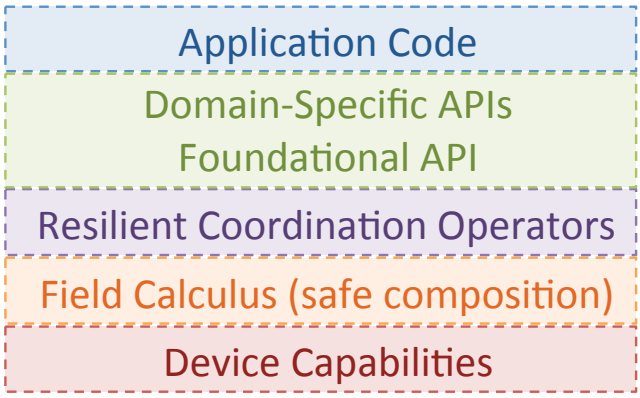

Figure 1. Aggregate programming stack: field calculus ensures safe composition and a well-defined mapping between aggregate specifications and local actions. On top of this foundation, higher level resilient building blocks are provided, and upon those operators a foundational API (and possibly domain specific libraries) can be written, ultimately easing the creation of complex distributed applications.

A system of composable self-stabilizing building blocks has then been constructed atop the field calculus [9]. Due to their nature, any program realised by composition of such blocks is provably self-stabilising (i.e., it converges to a stable value if the network remains stable long enough).

III. Protelis and the aggregate programming toolchain

The field calculus is a useful formal tool, but, as with most of other formal models, it is too low-level to be practical for software development without supporting syntactic sugar, tools, and libraries. Figure 1 shows a layered system of libraries addressing practical issues in order to make it easy to write application code. Protelis' goal is providing such tools: Protelis [7] is a programming language that implements the field calculus functional semantics, exposed using a syntax inspired by imperative style C-family languages in order to lower the effort required to learn the new language for those used to C, C#, Java, and similar programming languages. Protelis also provides a number of domain-specific APIs indirectly by being interoperable with Java, whose ecosystem contains thousands of libraries of all sorts.

No foundational API for resilient, situated, and distributed systems can be found in Java, however, nor to the best of our knowledge, in any other prior language or platform.

978-1-5090-6558-5/17 $31.00 © 2017 IEEE
DOI 10.1109/FAS-W.2017.186

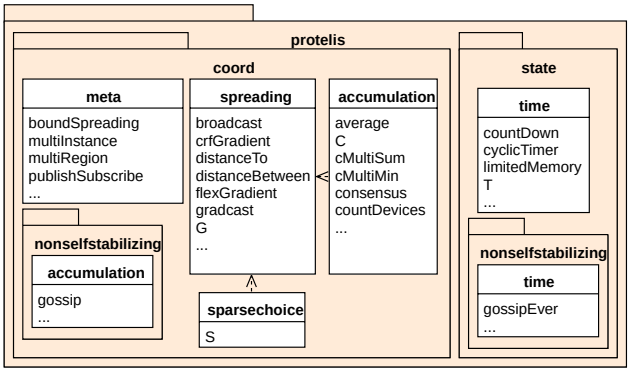

Figure 2. `protelis-lang` library structure. The goal of the library is to provide in a ready-to-use fashion many of the key algorithms used for resilience in distributed systems.

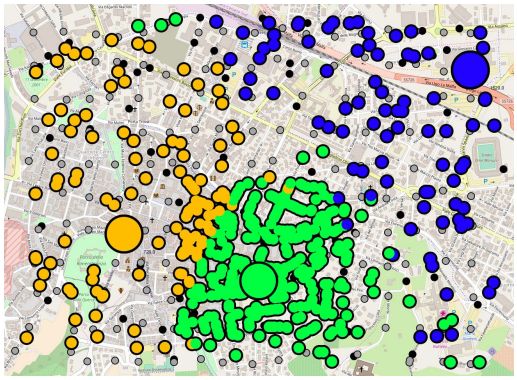

Figure 3. A screenshot of a typical Alchemist simulation of an in-development Protelis application executing on a real world map (in this case the Italian city of Cesena).

For this reason, a library called `protelis-lang` has been developed [8], by careful search and classification of the existing literature on self organisation and existing libraries and tools. The library, as depicted in Figure 2, contains algorithms for disseminating (`spreading`) and aggregating (`accumulation`) information, for partitioning the network (`sparsechoice`), for dealing with state and time (`state`), as well as algorithms that are less resilient but still useful in some situations (`nonselfstabilizing`), and a library of algorithms that leverage higher order functions to provide meta-functionalities, such as running multiple instances of an algorithms in parallel(`meta`).

Finally, one paramount issue to address when programming distributed systems is testing. Although a rich set of high level APIs greatly lowers the effort required to write programs, testing is always required during development both to verify the intended behaviour and produce performance assessments. Deploying the application and testing it in a real distributed setup is infeasible for many applications. For this reason, Protelis programs are often tested in simulation. Once simulations show the expected behaviour and performance match the required levels, the program can be deployed. Protelis itself does not embed any simulator, but its architecture makes it easy to link Protelis to existing simulation environment through Java. In particular, Protelis has been integrated with Alchemist [10], a simulator that is able to emulate a network of devices, possibly located in indoor environments or on real world maps, with support for GPS traces and movement along roads. A screenshot of an Alchemist simulated Protelis program is depicted in Figure 3.

IV. Future directions

Aggregate programming and Protelis have reached a level of maturity where it is possible to use them to define, study, and implement a wide variety of distributed and situated applications. As such applications are built, we anticipate that it will drive refinement and extension of the supporting libraries, as well as eventual development of a more general spatio-temporal operating system for the deployment, maintenance, and control of systems of interacting complex situated processes.

References

[1] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *IEEE Computer*, vol. 48, no. 9, pp. 22–30, 2015. [Online]. Available: http://dx.doi.org/10.1109/MC.2015.261

[2] D. Pianini, S. Dobson, and M. Viroli, "Self-stabilising target counting in wireless sensor networks using euler integration," *11th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2017, Tucson, AZ, USA, September 18-22, 2017*, 2017, to appear.

[3] M. Viroli, A. Bucchiarone, D. Pianini, and J. Beal, "Combining self-organisation and autonomic computing in cass with aggregate-mape," pp. 186–191, 2016. [Online]. Available: https://doi.org/10.1109/FAS-W.2016.49

[4] M. Viroli, G. Audrito, F. Damiani, D. Pianini, and J. Beal, "A higher-order calculus of computational fields," *CoRR*, vol. abs/1610.08116, 2016. [Online]. Available: http://arxiv.org/abs/1610.08116

[5] M. Viroli, G. Audrito, J. Beal, F. Damiani, and D. Pianini, "Engineering resilient collective adaptive systems by self-stabilisation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 2017, submitted for publication.

[6] J. Beal, M. Viroli, D. Pianini, and F. Damiani, "Self-adaptation to device distribution changes," in *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016*, 2016, pp. 60–69. [Online]. Available: http://dx.doi.org/10.1109/SASO.2016.12

[7] D. Pianini, M. Viroli, and J. Beal, "Protelis: practical aggregate programming," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, 2015, pp. 1846–1853. [Online]. Available: http://doi.acm.org/10.1145/2695664.2695913

[8] M. Francia, D. Pianini, J. Beal, and M. Viroli, "Towards a foundational api for resilient distributed systems design," *2016 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Tucson, AZ, September 18-22, 2016*, 2017, submitted for publication.

[9] J. Beal and M. Viroli, "Building blocks for aggregate programming of self-organising applications," in *Proceedings of the 2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, ser. SASOW '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 8–13. [Online]. Available: http://dx.doi.org/10.1109/SASOW.2014.6

[10] D. Pianini, S. Montagna, and M. Viroli, "Chemical-oriented simulation of computational systems with ALCHEMIST," *J. Simulation*, vol. 7, no. 3, pp. 202–215, 2013. [Online]. Available: http://dx.doi.org/10.1057/jos.2012.27