Spatial Computing in MGS

Antoine Spicher[1], Olivier Michel[1], and Jean-Louis Giavitto[2]

[1] LACL, Université Paris-Est Créteil,
61 av. du Général de Gaulle 94010 Créteil, France
{antoine.spicher,olivier.michel}@u-pec.fr

[2] UMR 9912 STMS – Ircam & CNRS, UPMC, INRIA
1 place Igor Stravinsky, 75004 Paris, France
jean-louis.giavitto@ircam.fr

Abstract. This short paper motivates and introduces the tutorial on MGS and spatial computing presented at UCNC 2012.

Keywords: unconventional programming paradigm, spatial computing, MGS, topological collection, topological rewriting, dynamical systems with a dynamical structure.

1 Spatial Computing

The notion of space appears in several application domains of computer science. Spatial relationships are involved in computer aided design applications, geographic databases, image processing... to cite a few. In these applications, space and spatial organization arise as the purpose or the result of a computation.

On the other hand, space can also play the role of a computational resource (*e.g.*, in parallel computation) or of a constraint to the computation itself (*e.g.*, in distributed systems).

Spatial Computing is an emerging research field that recognizes that space can be an input to computation or a key part of the desired result of the computation [6, 3]. Computations are performed in space and concepts like *position*, *distance metric* and *shape* matter. Space is then no longer an issue to abstract away, but a first-order effect that we have to make explicit in programs, to use, produce or optimize.

1.1 Spatial Computer in Physical Space

From the point of view of the computing devices, *spatial computers* are collections of local computational devices distributed through a physical space, in which: the interaction between localized agents is strongly dependent on the distance between them, and the "functional goals" of the system are generally defined in terms of the system's spatial structure (*e.g.*, formation control in robotics, self-assembly, etc.).

Not all spatially distributed systems are spatial computers. The Internet and peer-to-peer overlay networks may not in general best be considered as

spatial computers, both because their communication graphs have little relation to the Euclidean geometry in which the participating devices are embedded, and because most applications for them are explicitly defined independent of network structure.

Spatial computers, in contrast, tend to have more structure, with specific constraints and capabilities that can be used in the design, analysis and optimization of algorithms.

Systems that can be viewed as spatial computers are abundant, both natural and man-made, and blur the distinction between "space as a result" and "space as a resource". For example, in wireless sensor networks and animal or robot swarms, inter-agent communication network topologies are determined by the distance between devices, while the agent collectives as a whole solve spatially-defined problems like "analyze and react to spatial temperature variance" or "surround and destroy an enemy." Similarly, in biological embryos, each developing cell's behavior is controlled only by its local chemical and physical environment, but the eventual structure of the organism is a global property of the dynamic organization of the cellular arrangement.

1.2 Abstract Spaces in Computation

The elements of a physical computing system are spatially localized and when a locality property holds, only elements that are neighbors in physical space can interact directly. So the interactions between parts are structured by the spatial relationships of the parts.

However, even for non physical systems, usually an element does not interact with all other elements in the system. For instance, in a program[3], from a given element in a data structure, only a limited number of other elements can be accessed [11]: in a simply linked list, the elements are accessed linearly (the second after the first, the third after the second, etc.); from a node in a tree, we can access the father or the sons; in arrays, the accessibility relationships are left implicit and implemented through incrementing or decrementing indices (called "Von Neumann" or "Moore" neighborhoods if one or several changes are allowed).

Thus the interactions between the elements of a system induce a neighborhood relationship that spans an *abstract space*. We will show that the structure of the interactions has a topological nature: the set of elements can be organized through the interactions as an *abstract simplicial complex* [13] which is a spatial representation of the interactions in the system.

[3] The importance of *space in the computation process itself* has long been recognized, for example with the use of spatial relationships to structure and reason about programs; see [5] for an early reference.

2 MGS

MGS is an experimental declarative programming language [10, 12] used as a vehicle to experiment, to investigate, and to validate new concepts and tools for spatial computing. MGS relies on 3 ideas:

- a data structure is a field, or more precisely, a *topological chain*;
- a computation is a chain transformation;
- chain transformations can be specified using a new kind of rewriting.

A field is a classical notion in physics that associate a value to each point of a space (the temperature in a room is a field). A topological chain is a similar notion: it associates a value to the parts of a space built by gluing elementary parts following some constraints. The difference is that the underlying space is built by associating parts of various dimensions and is not restricted to be a set of points (points are elementary parts of zero dimension).

We use the term *topological collection* to stress the spatial view on data structures, and also because we relax some of the mathematical constraints used to give a nice algebraic structure to topological chains. Topological collections can be used to model a physical space or a logical one: we will give some examples where topological collections are used to model musical processes or to solve analogy, without referring to the actual euclidean space.

MGS embeds the idea of topological collections and their transformations into the framework of a simple dynamically typed functional language. Collections are just new kinds of values. MGS proposes several construction to build new topologies from existing ones and the rich type structure has proven useful to face various simulation problems in systems biology, chemistry, synthetic biology, etc.

Transformations are functions acting on collections. They are defined by a specific syntax using rewriting rules. They are first-class values and can be passed as arguments or returned as the result of an application. Such mechanism enables the definition of powerful *polytypic* operators [17], as for example the generic definition of discrete analogs of the differential operators used to manipulate fields in physics [16].

2.1 Computability *versus* Expressive Power

Topological collections and their transformation enable the unification in a same programming language of several biologically or biochemically inspired computational models, namely: Gamma [1] and the CHAM, P systems [20], L systems [21], cellular automata [27] and their variants. These models can be rephrased as the iteration of simple transformations on a topological collection; the difference coming from the topology of the space underlying the collection. However, we do not claim that we have achieved a useful theoretical framework encompassing the cited formalisms. We advocate that few notions and a single syntax can be consistently used to allow the merging of these formalisms *for programming purposes*.

The field of unconventional *computing* models, which is devoted to the study of the *complexity* of problems using a predefined set of (more or less exotic) basic operations, is not under focus here. The development of MGS is related to the field of unconventional *programming* models: MGS is used to study the *expressive power* of the spatial metaphor. The literature on programming language contains a wealth of informal claims on the relative expressive power of programming languages. However, this very notion remains difficult to formalize: for instance, we cannot compare the set of computable functions that a programming language can represent since nearly all programming languages are universal. As far as we know, there are only a few attempts to formalize this notion of expressiveness, see [7, 19]. These works mainly rely on the idea of translating a language into another, using a limited and predefined form of translation (if any translation is allowed, a universal language can be the target of the translation of any other one). However, these notions fail to explain why object-oriented languages (like `C++` or `Java`) are usually considered as more expressive than their imperative counterpart (like `C`).

The spatial metaphor has been proven useful at least at two levels in the development of new programming model. It is an heuristic to develop alternative mechanisms in programming languages or to invent new algorithms (for instance, a variety of successful established techniques for self-organization and self-adaptation arise from explicitly spatial metaphors, *e.g.*, self-healing gradients). The spatial metaphor proposes also new techniques originated in topology and geometry to specify and to analyze programs. These tools depart from the logical foundations of computer science and put forward alternative and complementary views in the nature of programming and computation.

3 Applications to (DS)²

In fine, unconventional programming languages have to be validated on real applications. A target application domain for MGS is the modeling and simulation of dynamical systems and especially those that exhibit a dynamic structure [10] (in short (DS)² for "Dynamical System with a Dynamic Structure"). This kind of dynamical systems is very challenging to model and simulate. For instance, many biological systems, *e.g.* in biological development, can be viewed as a dynamical system in which not only the values of state variables, but also the *set* of state variables, its organization and the evolution function, change over time. New programming concepts must be developed to ease their modeling and simulation.

Applications of MGS in the biological field are described for example in [22, 23, 2, 18, 26, 8]. The relations between MGS and the simulation of discrete dynamical systems are investigated in [25, **?**], the link between the MGS rule application strategies and stochastic simulation is sketched in [24].

MGS has also been used in other application area like self-assembly [15], autonomic computing [14] or automatic music analysis [4]. "Conventional" questions

still apply to "unconventional" program, and for instance, the model-checking of a small fragment of MGS is presented in [9].

All the examples presented during the tutorial are examples of actual MGS programs. The MGS interpreter is freely accessible from the MGS home page at `http://mgs.spatial-computing.org`.

Acknowledgements

We would like to express our gratitude to H. Klaudel, F. Delaplace and F. Pommereau at the Univ. of Evry, P. Prusinkiewicz at the Univ. of Calgary, and J. Cohen at the Univ. of Nantes for numerous discussions on biological modeling and formalization. We benefited from inspiring interactions with the unconventional computing community and especially S. Stepney at the Univ. of York, C. Teuscher at Portland State Univ., J. Durand-Lose at Univ. of Orleans, the membrane computing community with G. Paun at Univ. of Sevilla and M. Gheorghe at the Univ. of Sheffield. The chemical model of computation has been often challenging and we thanks J.-P. Bantre, T. Priol and P. Fradet at INRIA. Annick Lesne, R. Doursat, P. Bourgine and the french complex system community has provided a lot of motivations and insights. And great thanks are obviously due to the first supporters of the spatial computing movement: J. Beal at BBN, F. Gruau at Univ. of Paris South, S. Dulman and many others. This work has been funded by the CNRS, the Univ. of Evry, the Univ. of Paris-Est, Ircam, Inria and the ANR projects Autochem and SynBioTIC.

References

1. J.-P. Banâtre, P. Fradet, and D. Le Métayer. Gamma and the chemical reaction model: Fifteen years after. *LNCS*, 2235:17–44, 2001.
2. P. Barbier de Reuille, I. Bohn-Courseau, K. Ljung, H. Morin, N. Carraro, C. Godin, and J. Traas. Computer simulations reveal properties of the cell-cell signaling network at the shoot apex in Arabidopsis. *PNAS*, 103(5):1627–1632, 2006.
3. J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll. Organizing the aggregate: Languages for spatial computing. *CoRR*, abs/1202.5509, 2012. `http://arxiv.org/abs/1202.5509`.
4. L. Bigo, J.-L. Giavitto, and A. Spicher. Building topological spaces for musical objects. In *Mathematics and Computation in Music*, volume 6726 of *LNCS*, Paris, France, Juin 2011. Springer.
5. E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *Computing Surveys*, 3(2):67–78, 1971.
6. A. De Hon, J.-L. Giavitto, and F. Gruau, editors. *Computing Media and Languages for Space-Oriented Computation*, number 06361 in Dagsthul Seminar Proceedings. Dagsthul, `http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=2006361`, 3-8 september 2006.
7. M. Felleisen. On the expressive power of programming languages. *Science of Computer Programming*, 17(1-3):35–75, Dec. 1991.

8. J.-L. Giavitto. The modeling and the simulation of the fluid machines of synthetic biology. In M. Gheorghe, G. Paun, G. Rozenberg, A. Salomaa, and S. Verlan, editors, *Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers*, volume 7184 of *LNCS*, pages 19–34. Springer, 2011.
9. J.-L. Giavitto, H. Klaudel, and F. Pommereau. Integrated regulatory networks (irns): Spatially organized biochemical modules. *Theoretical Computer Science*, 431(0):219 – 234, 2012.
10. J.-L. Giavitto and O. Michel. Mgs: a rule-based programming language for complex objects and collections. In M. van den Brand and R. Verma, editors, *Electronic Notes in Theoretical Computer Science*, volume 59. Elsevier Science, 2001.
11. J.-L. Giavitto and O. Michel. Data structure as topological spaces. In *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, volume 2509, pages 137–150, Himeji, Japan, Oct. 2002. *LNCS*.
12. J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.
13. J.-L. Giavitto, O. Michel, J. Cohen, and A. Spicher. Computation in space and space in computation. In *Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *LNCS*, pages 137–152, Le Mont Saint-Michel, Sept. 2005. Spinger.
14. J.-L. Giavitto, O. Michel, and A. Spicher. *Software-Intensive Systems and New Computing Paradigms*, volume 5380 of *LNCS*, chapter Spatial Organization of the Chemical Paradigm and the Specification of Autonomic Systems, pages 235–254. Springer, november 2008.
15. J.-L. Giavitto and A. Spicher. *Systems Self-Assembly: multidisciplinary snapshots*, chapter Simulation of self-assembly processes using abstract reduction systems, pages 199–223. Elsevier, 2008. doi:10.1016/S1571-0831(07)00009-3.
16. J.-L. Giavitto and A. Spicher. Topological rewriting and the geometrization of programming. *Physica D*, 237(9):1302–1314, jully 2008.
17. J. Jeuring and P. Jansson. Polytypic programming. In J. Launchbury, E. Meijer, and T. Sheard, editors, *Advanced Functional Programming*, volume 1129 of *Lecture Notes in Computer Science*, pages 68–114. Springer, 1996.
18. O. Michel, A. Spicher, and J.-L. Giavitto. Rule-based programming for integrative biological modeling – application to the modeling of the λ phage genetic switch. *Natural Computing*, 8(4):865–889, Dec. 2009.
19. J. C. Mitchell. On abstraction and the expressive power of programming languages. In *TACS'91: Selected papers of the conference on Theoretical aspects of computer software*, pages 141–163, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.
20. G. Păun. From cells to computers: computing with membranes (P systems). *Biosystems*, 59(3):139–158, March 2001.
21. G. Rozenberg and A. Salomaa. *Lindenmayer Systems*. Springer, Berlin, 1992.
22. A. Spicher and O. Michel. Using rewriting techniques in the simulation of dynamical systems: Application to the modeling of sperm crawling. In *Fifth International Conference on Computational Science (ICCS'05), part I*, volume 3514 of *LNCS*, pages 820–827, Atlanta, GA, USA, May 2005. Springer.
23. A. Spicher and O. Michel. Declarative modeling of a neurulation-like process. *BioSystems*, 87(2-3):281–288, Feb. 2007.
24. A. Spicher, O. Michel, M. Cieslak, J.-L. Giavitto, and P. Prusinkiewicz. Stochastic p systems and the simulation of biochemical processes with dynamic compartments. *BioSystems*, 91(3):458–472, March 2008.

25. A. Spicher, O. Michel, and J.-L. Giavitto. A topological framework for the specification and the simulation of discrete dynamical systems. In *Sixth International conference on Cellular Automata for Research and Industry (ACRI'04)*, volume 3305 of *LNCS*, Amsterdam, October 2004. Springer.
26. A. Spicher, O. Michel, and J.-L. Giavitto. *Understanding the Dynamics of Biological Systems: Lessons Learned from Integrative Systems Biology*, chapter Interaction-Based Simulations for Integrative Spatial Systems Biology. Springer Verlag, Feb. 2011.
27. J. Von Neumann. *Theory of Self-Reproducing Automata*. Univ. of Illinois Press, 1966.