Chapter 16
Organizing the Aggregate: Languages for Spatial Computing

Jacob Beal
Raytheon BBN Technologies, USA

Stefan Dulman
Delft University, The Netherlands

Mirko Viroli
University of Bologna, Italy

Nikolaus Correll
University of Colorado Boulder, USA

Kyle Usbeck
Raytheon BBN Technologies, USA

ABSTRACT
As the number of computing devices embedded into engineered systems continues to rise, there is a widening gap between the needs of the user to control aggregates of devices and the complex technology of individual devices. Spatial computing attempts to bridge this gap for systems with local communication by exploiting the connection between physical locality and device connectivity. A large number of spatial computing domain specific languages (DSLs) have emerged across diverse domains, from biology and reconfigurable computing, to sensor networks and agent-based systems. In this chapter, the authors develop a framework for analyzing and comparing spatial computing DSLs, survey the current state of the art, and provide a roadmap for future spatial computing DSL investigation.

INTRODUCTION
Computation has become cheap enough and powerful enough that large numbers of computing devices can be embedded into nearly any aspect of our environment. A widening gap, however, exists between the potential users of embedded systems (biologists, architects, emergency response teams, etc.), and the increasingly complex technology available. Typically, the users know what they want from the aggregate of devices, but the programming or design interfaces that they are presented with operate mainly at the level of individual devices and their interactions. Thus, biologists who want to monitor wildlife with mesh-networked sensors end up having to debug real-time code for parallel algorithms in order to minimize Joules-per-packet, and architects who want to create responsive buildings end up worrying about how to create distributed algorithms.

DOI: 10.4018/978-1-4666-2092-6.ch016
Organizing the Aggregate

Figure 1. A widening gap exists between the application needs of users and the implementation details of those applications on increasingly complex systems of interacting computing devices. This gap must be crossed by global-to-local compilation: the transformation of specifications of aggregate behavior to actions of individual devices.

to ensure correct time-synchronization across the application modules.

Similar gaps are found in many other areas besides embedded systems. For example:

- Robots are becoming cheaper and more reliable, such that teams of robots can now be used in many more applications. For instance, search-and-rescue workers might use a group of unmanned aerial vehicles to search a wilderness area. These search-and-rescue workers should be able to specify aggregate behaviors like sweep patterns, rather than worry about how to share localization and progress data in order to keep those sweep patterns consistent between robots. Modular and reconfigurable robotic systems have similar issues.

- Reconfigurable computing devices like Field Programmable Gate Arrays (FPGAs) can solve complex problems extremely quickly. Programming and code reuse are difficult, however, because subprograms need to be tuned for their layout on each particular model chip and their context of use relative to other subprograms. Similar issues are likely to emerge in multi-core systems as the number of cores continues to rise.

- In synthetic biology, it will soon be desirable to program engineered biological organisms in terms of the structure of tissues or colonies, and of their interaction patterns, rather than by selecting particular trans-membrane proteins and adjusting their signal dynamics and metabolic interactions. Emerging chemical and nanotechnological computing platforms are likely to face similar issues.

Fundamentally, this gap lies between programming the aggregate and programming the individual (Figure 1). For the users, the application is often best specified in terms of collective