SPACE-TIME PROGRAMMING
2012 Research Statement
Jonathan Bachrach

The computational landscape is drastically changing. Processing is becoming virtually free, and sensors and actuators are becoming embeddable and affordable, allowing us to manufacture myriads of computational devices and embed them in the world. Unfortunately, as computing becomes smaller and more plentiful, it becomes harder to make parts error free and calibrated. Furthermore, communication delays are becoming sizable, forcing devices to directly communicate with and directly affect only local neighbors. In short, the assumptions that data is both perfect and available are no longer true. Traditional engineering approaches do not apply to this computing class of cheap, distributed, possibly faulty, locally communicating devices, that we call “spatial computing”. We are forced to radically rethink our computing models, languages, and practices.

Spatial Computing opens up a number of exciting platforms that have the potential to embody unparalleled grace, fidelity, and pervasiveness. Some example platforms are kinetic structures, sensor networks, swarm and modular robotics, peer to peer networks, reconfigurable computing, cloud computing, and biofilms. Application areas include search and rescue, threat avoidance, target tracking, distributed energy management, programmable matter, data search, synthetic biology, and active structures. Furthermore, because of power and speed tradeoffs at current and future transistor sizes, even conventional and enterprise computing are embracing very large scale parallel processing requiring novel programming approaches to utilize their potential.

The goal of my research is to create organizational principles and high level programming tools for Spatial Computing, which I call “space-time programming”. We look to biology for inspiration, where we can find many examples of robust engineering of Spatial Computing systems. While we can learn from biology about many mechanisms, we still need a practical engineering discipline, one that spans from high level specifications to implementation. Borrowing and stitching together biological mechanisms would be unworkable and direct programming of device level rules would be unwieldy. We instead strive to construct application level, intentional and intuitive program specification that are amenable to translation into device level rules.

Our basic approach is to program the devices as a single spatial entity which fills the space through which the devices are scattered and to develop a series of high level and scalable abstractions, modules, and compositional mechanisms that lead to robust, predictable and scalable software. We look to engineering and computer science for techniques for managing complexity in the design of complex systems. Software applications greatly benefit from being decomposed into levels of abstraction and hierarchies of modules with explicit interfaces, contracts and data structures at each level. Abstractions and modules can be built and tested independently, they can be named, interface compatible modules can be interchanged, and from there code reuse can be enhanced, and a separation of concerns can be maintained. In short, resulting applications are less complex, having fewer moving parts, and, as a consequence, are smaller, more reliable and faster to develop. Machine learning and program language research have until recently developed independently, whereas I would like to consider hybrid approaches where adaptive behaviors are guided by language expressions. Examples of self adaptation are adaptive routing, self repair, and homeostasis. Languages force programmers to make certain things explicit and gain great power by hiding other details, details hidden behind a capable runtime. Traditional dynamic languages have hidden memory management operations for example. But I am interested in pursuing languages that involve much more powerful runtimes that expose robust interfaces to their self adaptive substrate.

Such higher level languages will have the technical advantages that high level descriptions of machine behavior will get compiled down to low level operations that are implemented on individual components. This approach provides machine descriptions that are more natural, powerful, and able to meet application requirements, and also provides scalable and robust software solutions that are independent of precise hardware details. I also plan to pursue more advanced languages that allow the direct statement of task goals, constraints, and models and with their runtimes performing planning, constraint satisfaction, and failure diagnosis. These approaches show great promise in the future where systems are more complicated, longer lived, and/or run unattended. For example, in a good choreographic language for modular robotics, one would specify target postures in abstract spatial terms. A spherical posture for a kinetic structure could be specified and components would then conspire to occupy equidistant spherical surface points. This approach would have the advantage that the abstract descriptions would be tolerant to the addition and deletion of components as well as defects and initial conditions. From a programmer's point of view, this would be incredibly liberating since forms could be developed independently of the kinematic models necessary to control their behavior. On a technical level, distributed algorithms would be one solution to developing a whole new vocabulary of space, time, and posture in order to describe robust and flexible ensemble behaviors.

One key abstraction that we have used is the "amorphous medium" which allows the programmer to factor the problem into programming continuous space and time and approximating continuous space and time on a discrete network. Programs can be written in terms of continuous space and time, while the compiler/runtime maps to actual physical realizations in terms of discrete computational devices at specific points. The key advantage is that the programs can be written once and work on a variety of actualizations with varying resolutions, while the runtime adapts to device birth, death, failure, and movement.

We have developed a language called Proto which embodies this amorphous medium abstraction and permits the construction and composition of high level modules. Programs written in Proto are succinct -- often two orders of magnitude smaller than programs written in more traditional languages. The compiler utilizes aggressive type inferencing and optimization yielding tiny and efficient runtimes with known space requirements allowing us to target extremely impoverished runtimes. Proto has been implemented as a simulator with extensive visualization and development abilities and ported to several physical platforms including sensor networks, swarm robots, and modular robots. The general approach has been widely applicable to a number of Spatial Computing problem domains. We have replicated results from the MIT Amorphous Computing group and have developed new robust algorithms including a self healing active gradient, a new time synchronization algorithm, and a new leader election algorithm, among others. Furthermore, we have started to model biological phenomena such as slime molds and mammalian morphogenesis with promising early results. Finally, Proto has been quite well received outside of MIT, leading to my being invited to be on a 2007 DARPA ISAT study on Engineered Ensemble Effects, several lectures at Harvard, an invitation to NDIST 2007 workshop on engineered emergence, a keynote address to VIPSI-2008 Slovenia, and an NSF bio inspired computing grant (of which I was the primary author and editor).

While Proto has concentrated on distribution, viral programming, and small footprint, other efforts have focused on additional aspects of the space-time programming vision. We have pushed upwards towards higher abstractions while exploring the scripting of multimedia in the language called Gooze, concentrating on bulk operators, temporal abstractions, and targeting GPUs and SIMD coprocessors. In the language Gel, we have concentrated on drilling down, targeting more constrained devices such as FPGAs (with an eye towards biological cells) requiring more detailed modeling of space and time and producing hardware oriented abstractions.

Gooze allows artists and scientists to create and script multimedia effects modules combining wide ranging techniques such as 3D, vector, particle, and image based graphics. Stream processing style languages raise the abstraction level for time-oriented domains where time abstractions are provided and time based operations can be conveniently specified and powerfully composed. The concise and stream-based nature of the language has been invaluable for experimenting with interactive multimedia and computer vision algorithms and goes far beyond typical visual programming approaches. Targeting GPUs allows the effects to make use of supercomputing class resources and opens up great opportunities in scientific computing. Finally, Gooze has been used in many large scale professional art installations and performances around the world.

Gel is a new high level hardware scripting language that makes it convenient to specify hardware designs and permits the construction of paradigm specific libraries. It features a rich set of data types, is extremely succinct, and is expression oriented. Modules are described as functions and composed through function calls. Types and bit widths are inferred automatically and functions are automatically folded and common subexpressions are identified. A compiler has been developed that translates Gel to Verilog and a number of applications have been developed and compiled down to actual FPGA devices. Finally, we have developed a postcard sized FPGA board with audio/video IO as a first demonstration vehicle.

Because languages provide the representational substrate for Spatial Computing, radical language exploration is critical to the future of this new field. Spatial Computing opens up exciting new possibilities for understanding Biology and engineering robust and scalable systems with spatial extent, but requires a new architectural approach and set of programming models. It will be the focus of my research to invent these approaches and models through technical investigations, and to apply them to help create a new class of computational substrate and a new engineering discipline.

Developing new languages must go hand in hand with aggressive exploration of new platforms combined with biological modeling and application development. Through technical and scientific investigations, and collaboration with leading technologists and scientists, my research will focus on the invention of novel hardware and software for Spatial Computing, and will apply them to help create a new approach to engineering these systems. I am uniquely positioned and poised to significantly contribute to the new field of Spatial Computing, having already produced three significant languages, several hardware platforms, published eight refereed papers in this field, and have a large research pipeline. Computer Science departments have historically been environments where technological and scientific innovation have introduced new computational disciplines to engineering and scientific communities. I look forward to being part of this ongoing history by coupling programming models and Spatial Computing systems to expand the computing and scientific landscape.