Spatial Computing for non-IT Specialists

Steffan Karger, Agostino Di Figlia, Maurice Bos, Andrei Pruteanu, Stefan Dulman
Delft University of Technology, the Netherlands
{s.j.karger, a.difiglia, m.bos-1}@student.tudelft.nl, {a.s.pruteanu, s.o.dulman}@tudelft.nl

ABSTRACT

Designers and architects are showing an increasing interest for intelligent and interactive building environments, employing large numbers of networked embedded devices, often equipped with wireless communication capabilities. Building small prototypes is usually feasible with a central-control approach. As soon as the prototype needs to be scaled up in the commissioned buildings, complexity arises due to the large number of interacting devices.

In this paper, we link the interactive environments applications with the field of spatial computing. As we will show, the two are strongly correlated and spatial computing can prove to be an elegant solution for the problem at hand. Moreover, spatial computing has the potential of uncovering new designs, based on the emergent behavior properties of large-scale networks. We propose a new framework, called IDS (Interactive Design Studio), which allows for exploration of new design possibilities employing networked embedded systems, without the expertise of IT-specialists.

The IDS framework is built on top of the Proto programming language and targets the protoDeck interactive floor. We showcase its capabilities via two application scenarios and confirm its benefits by means of a survey involving architecture students. Finally, we show implementation details of the complete software stack and experimental results from deployment on the embedded platform.

Keywords

spatial computing, distributed systems, interactive design, embedded systems, software framework

1. INTRODUCTION

Recent years have seen an explosion in the number of networked devices embedded into engineered systems. Wireless sensor networks, swarming robots, mobile ad-hoc networks, smart phones and smart appliances are just a few well-known application domains where this has already become a reality. Properties such as flexibility and ease of use make networked systems attractive solutions for problems outside the information technology domain.

Architects show an increasing interest for intelligent and interactive building environments [7]. Current state-of-the-art includes designs such as the *Ada* floor [8]. It is composed of interconnected tiles capable of interacting with the users stepping on it by means of light patterns. Another example is the *Healing pool* by Brian Knep[15], consisting of a projection of organic patterns on the floor; the patterns self-heal after being torn apart by people walking around. These projects emphasize the growing trend of designing complex interactive spaces in private or public buildings [11, 18].

Even though interactivity is achieved with different technologies, a common property is occurring in all applications: computational elements are spread out and fill the design space. They interact with each other and the users in various ways leading to complex behaviors. When surveying the current deployments, we noticed that the current installations usually employ some form of centralized control. For prototypes consisting of a small number of elements, that is not an issue. When scaling up to large setups, centralized control becomes almost impossible and the distributed interaction is simply dropped. When trying to mimic distributed systems, such as in the case of the healing pool application [15], the technology used (projectors, cameras and image recognition, a limited-sized deployment area) implies the need of a specific, carefully controlled environment, considerably limiting the design freedom.

At high level, we believe that the problem architects face when *designing interactive environments* is very close to *the killer application* for the field of *spatial computing*. The correlation between the two topics is obvious and we can break down the application scenario in two parts linked to the bottom-up and top-down design of complex systems:

- With an ever increasing number of computing devices equipped with sensing and actuation capabilities, there is a quest for exploring feasible and interesting interactive designs that make use of embedded platforms. Non-IT specialists need ways to fast prototype ideas on large-scale systems, while abstracting from the underlying technological complexity related to communication protocols, programming languages, operating systems, embedded virtual machines, hardware platforms etc.

- Secondly, the complexity that arises in such distributed systems, in the form of top-down translation of specifications for system behavior into local rules (also called global-to-local compiling) is a challenging research question that has been addressed before for different application domains [17].

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

Both problems are still open for research in the field of complexity theory in general and spatial computing in particular. To the best of our knowledge, solutions to both problems include human expertise [9]. We do not hold a completely autonomous solution to these problems - we merely attempt to provide a framework in the form of a software tool chain that makes use of distributed computing, sensing and actuation. The framework targets designers and architects - the non-IT specialists - and aims to help them explore various interactive design ideas via spatial computing constructs.

Previous attempts of specifying global system behaviors via spatial computing constructs were targeted at the so-called IT specialists: we refer the reader to a number of spatial computing domain-specific languages (DSLs) made available in recent years, such as Proto [3], Kairos [10] and TOTA [16]. Our framework is built on top of such a programming language, Proto. We further elaborate on this in Section 2.

The framework we present in this paper has been tailored for protoSpace [12] at TU Delft, Faculty of Architecture, Hyperbody Group[6]. The space has an interactive floor, protoDeck, consisting of 189 tiles each equipped with a microcontroller, RGB leds and a pressure sensor (Figure 1). Due to the power requirements of the LED's, the nodes are powered from the grid. ProtoSpace 3.0 [12] also comprises other multimedia devices such as beamers, a complex sound system and various interactive objects. The ambition is to use protoSpace and all its components as an ecosystem capable to create interactive user experiences. To achieve that, we provide the non-IT specialists with a friendly design tool chain. It facilitates the design of interactive spaces for various events such as art exhibitions, dance performances, teaching activities, social events, etc.

The design tool chain (Figure 2) comprises four components: GUI, StateChart Compiler, DeckSim and protoDeck. They correspond to the four stages of the design process. The GUI serves as a graphical specification tool that eases the description of the tiles' behavior. The GUI produces a state chart representation of the behavior and is given as an input to the StateChart Compiler which generates the platform specific code for DeckSim and the protoDeck hardware. The SC Compiler aims to substitute the embedded systems specialists in the design loop.

The paper has the following outline. In Section 2 we discuss related work for both interactive spaces and spatial computing platforms. Section 3 outlines and describes the framework and its components in detail. An example scenario is given in Section 4. We show the experimental results in Section 5. We discuss the results in Section 6. Finally, we conclude in Section 7.

2. RELATED WORK

Interactive environments have become popular in recent years [5] and several aspects achieving interactivity have been explored. Next section will discuss three interactive spaces (*Ada* [8], *Healing Pool* [13] and *Hallway monitoring* [2]) and their main characteristics in terms of interactivity type and adopted techniques.

2.1 Interactive Environments

Delbruck et al. have created a tactile luminous floor, *Ada*, for an interactive autonomous space. The space consists of a floor, projection screens, microphones, ceiling cameras, speakers and theatre lights. The tiles on the floor are equipped with tactile load sensors and RGB lamps. They are networked as a cellular automata using an industrial automation network, Interbus. A centralized approach is used for controlling the floor's behavior. In fact, the tile's local controller delivers the data to a PC which controls the behavior. In contrast, our approach aims to provide a complete distributed approach to achieve user interactions.

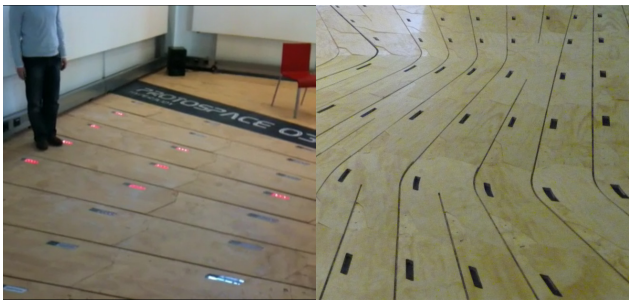


Figure 1: The protoDeck floor. On the left the floor detecting presence of a person, on the right an impression of the shape of the floor.

Another example of an interactive floor is *Healing Pool* [13]. It was presented at the exhibition in the Brauer Museum of Art (Valparaiso University). The *Healing Pool* is an interactive video installation equipped with video projectors, cameras, custom software and a vinyl floor. The main characteristic is the ability to project organic patterns that are torn apart by visitors walking on the floor. Ultimately, they rebuild themselves in an always unique way. Even though the work relies upon artificial intelligence and imaging techniques it shows the strain and increasing interest in interactive spaces. We believe that large-scale complex interaction can be achieved only by means of distributed systems of sensors and actuators via the spatial computing paradigm.

An approach technologically more similar to ours is Hallway Monitoring [2]. In this project, wireless sensor nodes have been placed underneath a hallway floor. The sensor nodes are able to sense pressure on the tiles and actuate lights and speakers on the hallway walls. Due to the limited space and the lack of direct feedback from the tiles, the possibilities for complex interaction are also limited. No extra objects to interact with can be placed in the hallway and the movement of a person is unidirectional only. The set-up offers interesting research possibilities from the computer science viewpoint, but it lacks expressiveness for designers and architects.

2.2 Spatial Computing Platforms

The goal for the protoDeck space is to have an interactive prototyping platform in which architects and designers can develop interactive environments. The spatial and temporal properties are both considered as fundamental constituents. This is strongly correlated to the Spatial Computing paradigm, which endeavors to unleash the potential of using the notions of space and time in programming of distributed systems. In the past, several efforts were taken in this area [4]. In this section we will discuss three of them (Proto [3], Kairos [10] and TOTA [16]). Additionaly, we explain why we chose Proto for this project.

Proto[3] is a functional language that employs the concept of an amorphous medium abstraction[1], in which the discretization of space and time is hidden from the end user. When using Proto, programs do not incorporate their own algorithms for communication and communication related services (e.g, neighborhood discovery or distance estimation). The information about the network and neighborhood is presumed to be available and should be taken care of by the underlying layers. These features enable Proto programs to be very compact. Proto comes with a tool chain that includes a compiler, a simulator and a virtual machine.

Kairos[10] is based on ideas from shared-memory parallel programming. It delivers three primitives: a node abstraction, delivering the programmer tools to manipulate (lists of) nodes, a list of one-hop neighbours and remote data access. Remote data access does not guarantee delivering the correct value, instead Karios relies on 'eventual consistency'. Eventually the system should converge to the correct solution to the problem at hand. While executing tasks, Kairos blocks the execution of the application. Kairos' functionality is delivered through an API, which can be accessed from imperative programming environments. Kairos still remains in a proof-of-concept state.

TOTA[16] stands for 'Tuples over the Air'. It is based on the notion of Tuple fields, which can be seen as information fields from nature, like force fields or chemical gradients. Tuples consist of a content element, a propagation rule and a maintenance rule. Tuples are produced locally and then distributed through the network. Its limitation comes from the fact information from tuples can not be aggregated. TOTA exposes a Java API to the end user.

When choosing a platform we have to remember our goal: an easy to use environment for architects and designers. For this, we need to be able to generate programs from a graphical representation (in our case a state chart) and a tool chain that is feature-complete. The translation from state charts to Proto code is a viable option.

3. SYSTEM DESCRIPTION

The proposed framework is described by the block diagram in Figure 2. The diagram shows four components that correspond to the four stages of our design process. In the following we briefly describe each component and the design rationale behind it.

3.1 GUI

The user interacts with the GUI which consists, in the current state, of a graphical state chart editor. It allows non-IT experts to design a state chart representing the desired behavior of individual tiles. The state chart describes the state transitions of a single tile of the protoDeck floor. The ultimate ambition is to design a user friendly and easy to use GUI for specifying system level and node level behaviors that will hide the cumbersome design of a state chart. The GUI produces an xml file that is structured according to the W3C State Chart extensible Markup Language which serves as an input to the SC Compiler. The rationale behind the use of a state chart representation for the tile's behavior is the following. Since the system under design is reactive and its elements are connected in a mesh topology, the state charts proved to be suitable modeling technique. Moreover, state charts have a way, though limited, of specifying time which is sufficient for our application purposes.

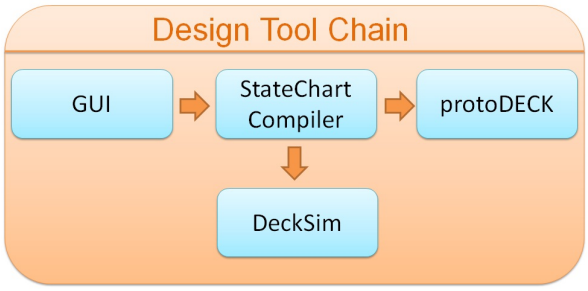


Figure 2: Framework block diagram

3.2 StateChart Compiler

StateChart Compiler is a java based tool that parses the scxml file and produces code for a specific platform or environment. In our case the two supported languages are *Netlogo*[19] and *Proto*[3]. In order to be able to support as many end platforms as possible the specification of the scxml presents a strongly generalized set of events, conditions and actions that can easily be mapped to any specific platform code. In our specific case, the end platforms are DeckSim and protoDeck. The former is a Netlogo based simulation environment, while the latter consists of a mesh network of embedded system devices running the *DelftProtoVM*. The compilation process is a customizable process which receives as input a configuration file describing the used hardware platform in terms of its sensors and actuators and ,in addition, it uses the language specific spatial computing libraries. The SC Compiler interprets the scxml by translating the state chart with the provided hardware specifics. While, the spatial actions are mapped by referencing the provided language specific library. The SC Compiler can be further extended by adding a desired new language specific library and translator module.

3.3 DeckSim

DeckSim is a Netlogo based simulator that provides the possibility to test and have visual feedback of the state chart behavior diagram. The simulation models protoDeck behavior and allows to simulate and test interactions. This way a speedup of the design process can be achieved. The iterative design process consists of a design and test cycle that is usually performed by a designer or architect during sketching or prototyping. They are able to iterate from the specification phase to the test phase and back before deploying the code to protoDeck. Simulations can be run either stand alone or guided.

3.4 Embedded Software Platform

3.4.1 DelftProto VM

The DelftProto VM is a virtual machine that executes Proto bytecode. In September 2011 it replaced the original virtual machine in the Proto distribution, the 'Proto Kernel'. The DelftProto VM code is written to be extremely portable; we were able to succesfully run it on ARM Cortex, Atmel ATmega, MSP430, Intel 586 and AMD 64.

The instruction set of the VM is designed specifically for spatial computing applications. It incorporates instructions that form an aggregate from neighborhood information and common (high level) data types such as vectors are natively

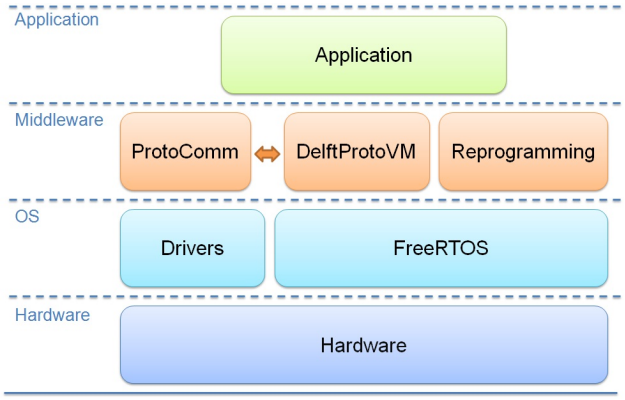


Figure 3: Schematic view of platform components.

supported. Most instructions have implicit operands and work on multiple data types, which allows complex programs to be compiled to very small binaries that can be executed by the VM. For example, a simple gradient algorithm has a size of 35 bytes.

An improved virtual machine based on the DelftProto VM, the *Delft VM*, is currently being developed. Whereas the DelftProto VM is specifically made for programs written in Proto, the Delft VM supports other languages as well. The instruction set makes it easier to generate code from an imperative language, although the focus still lies on functional languages.

3.4.2 Communication and Scheduling

When an application is translated to local rules, it is compiled to run on the virtual machine. We provide an embedded software framework that is easily portable to different hardware platforms. A schematic view of the building blocks is shown in Figure 3.

We use protoDeck as prototyping platform(Figure 1). Each tile is equipped with RGB leds and a pressure sensor. The nodes beneath the tiles are based on NXP LPCXPresso LPC1769 (ARM Cortex-M3) modules connected in a wired mesh configuration. Inter node communication uses the chip's UARTs at 115K2 baud, but our implementation is built to be easily adapted to other communication methods. A wireless (2.4 GHz, 802.15.4 based) version is planned for future experiments. The floor is able to interact with other objects in the room, for example tables, chairs, external lighting and beamers.

The first software layer consists of the FreeRTOS operating system and hardware-specific driver libraries. These deliver basic facilities for the layers on top. The middle layer consists of three parts: the communication library called ProtoComm, the DelftProto VM and a reprogramming facility to update Proto applications virally.

The ProtoComm library supplies the VM with neighborhood information in a best-effort way, since achieving perfect knowledge of all neighbors is generally not possible in real world applications. It takes care of neighborhood discovery, distance estimation, lag estimation, exchange of state information and application updates.

ProtoComm is designed to be compatible with multiple communication types. Incoming data is buffered by device driver interrupt routines. ProtoComm scans the buffers for valid packets and processes them. Packet processing that involves changing the state of the virtual machine is postponed until a virtual machine execution round is completed.

The reprogramming library enables the user to virally roll out new Proto applications without the need to update each node manually. Applications consist of (compact) Proto bytecode, what makes updating the application easier and faster compared to updating a complete platform binary. Nodes keep track of their application version and automatically disseminate new applications as soon as a new version is detected in the neighborhood. An update process is initiated by updating a single node with the new application.

For both disseminating state information and application updates, a negotiation based approach (ADV-REQ-DATA) such as in [14] is employed to avoid broadcast storms and hidden terminal problems. For the case of state updates, which are just exchanged between direct neighbors and thus not propagated, we can reduce completion time by replacing the first advertisement after a detected change in local state with a data packet. The negotiation based technique continues to run in the background to take care of the occasional failed initial communication.

4. INTERACTIVE SCENARIO

As an example scenario we propose using protoSpace to enhance art exhibitions. We imagine the space consisting of several interactive components such as the protoDeck and responsive furniture which engage the visitor and guide him through the various art objects. When visitors walk across the room, the floor leaves a colored trail along the visitor's path. The art objects are placed in showcases which are demarcated by the floor by creating a pulsating light circle around them. Whenever a visitor approaches one of the showcases it triggers an increase of the circle's radius which will surround object and visitor. The light patterns will change triggered by different factors such as the number of people that are close to an art object or the crossing of different visitor trails.

The aforementioned scenario can be composed of several sub-behaviors the space performs in a distributed fashion. Such sub-behaviors are, for example, a distance metric or a desired light pattern. For that reason, we divide the scenario in several sub-behaviors. At the current stage of our project, we performed our experiments focusing on two test applications - a gradient application and firefly synchronization algorithm. By creating a gradient we were able to define a distance metric and show the viability of using spatial primitives. Firefly synchronization is a suitable test to assess the viability of using time primitives while continuously stressing the communication layer.

5. EXPERIMENTAL RESULTS

5.1 User Survey

In order to confirm the benefits of IDS we performed a survey amongst architecture students that have used protoSpace in one of their projects. We were interested in three different topics. Firstly, what kind of network of many 'smart' components they would like an environment to be equipped with (Figure 4a). Secondly, we asked what they would like to improve or upgrade in protoSpace (Figure 4a). Finally, we asked what kind of software tools they would like

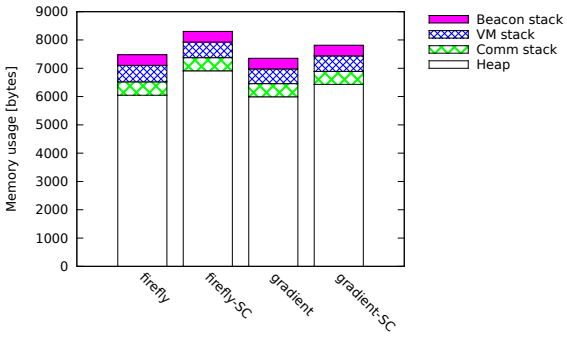


Figure 5: RAM usage comparison for the four Proto applications.

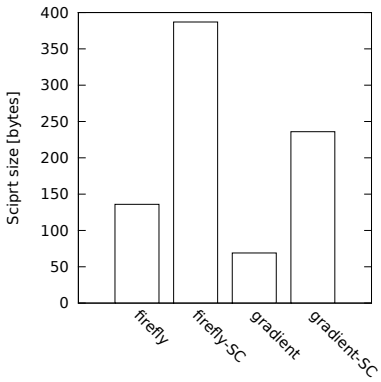


Figure 6: Script size comparison for the four Proto applications.

to have at hand to design or prototype protoDeck or alike (Figure 4c).

In Figure 4 we show the radar charts of the survey outcome. In Figure 4a we can see that the most requested type of 'smart' component is a network of smart furniture followed by a light control system and control software for the space. Smart furniture is for example a chair, table or other object equipped with sensors and actuators. Figure 4b shows the most requested features to add to the existing prototype. These are a designer community, crowd sourcing and more advanced lighting capabilities. Community and crowd sourcing are in a certain way related. With community the students mostly intend a forum like website where they can discuss current and past projects regarding protoSpace. With crowd sourcing we mean code sharing. Finally, Figure 4c shows that most of the surveyed students would like to have a simulation tool or a web based application that is capable to simulate protoSpace. The gathered information is used to guide the ongoing research.

5.2 Simulations versus Testbed Evaluation

As most embedded software developers will acknowledge, the step from simulation to deployment is far from trivial. In this section we highlight several issues we came across while making this step.

Most simulations are based on *unrealistic assumptions*, like instant communication, infinitesimal computation times and a certain level of synchrony. In the actual system usually they do not hold. Our specific approach to handle this is still work in progress.

A larger number of nodes implies a higher risk of *hardware failures*. We conducted some experiments with the previous generation of protoDeck nodes. The testbed consisted of 24 nodes (3 by 8 mesh) with wired serial connections. The connections were not amplified and cables were connected directly to pin headers. This set-up was working when faced with a number of hardware failures. People walking on the floor caused wires to loose contact or suffer from breakage and caused some nodes to die. The high number of components made these failures a rule rather than an exception. This calls for software that can handle failing nodes, unreliable communication.

There are possible *software failures* that will not show until the software is run on the actual embedded platform. However, once on the hardware, it is much more difficult to find the cause of the failures. Getting the simulated environment closer to the real world scenario would help to discover failures earlier.

5.3 Memory Usage and Script Size

To asses *IDS*, we implemented two spatial computing applications, *building a gradient* and *firefly synchronization*. Both are implemented as a manually crafted Proto application as well as an application generated by IDS. Tests are run in a five node network in which one of the nodes was connected to three neighbors. Memory usage statistics were collected from this node.

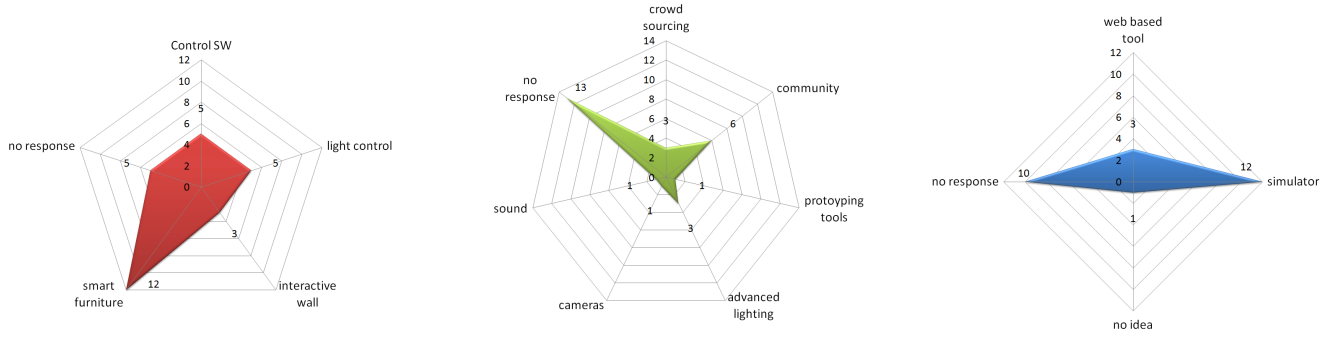
Shown in Figure 5 is the maximum memory usage in bytes for the various applications. As can be observed, the bulk memory is consumed by the heap. The heap is used for storing incoming packets and during execution of the virtual machine. Our primary interest here is the increase in memory usage with increasing application complexity, not the absolute memory usage. When considering script size, shown in Figure 6, as a measure for application complexity we can observe that the increase in memory usage is approximately 10% when the application complexity more than doubles. This indicates there is space for implementing much more advanced applications.

Also note that although the manual and state chart versions of applications are functionally equal, there is a notable difference in script size. This is an indication that there is much to gain from further optimization within the Proto compiler. A compiler should ideally be able to reduce the state chart version of an application to the same script size as the manual version.

6. DISCUSSION

The trend in designing interactive environments is the driving force for creating a toolchain to ease the development of interactive spatial computing applications. To confirm this, we conducted a survey amongst designers and architects. The survey confirms our expectations on the interest in such environments and the need for a simulator to enable fast prototyping. During design toolchain testing, we realized that the current GUI requiring the use a state chart representation is still a cumbersome for designers or architects. Therefore, another level of abstraction is needed to hide the state chart representation.

In an effort to validate the viability of spatial computing



(a) 'Smart' architectural components (b) Requested features (c) Desired tools

Figure 4: Survey results

for interactive environments, we implemented a toolchain that is capable of designing an application using state charts, testing the application in a simulator and running it on actual hardware. A proof of concept implementation of the spatial computing primitives confirms the validity and shows the viability of this approach. For larger applications there might be a need for further optimization of the memory consumption.

7. CONCLUSIONS AND FUTURE WORK

In this paper we introduce a software platform called *IDS* that uses the concepts of *Spatial Computing* to facilitate to non-IT specialists the fast-prototyping of interactive designs using distributed embedded systems installations. It is able to translate high-level specifications into agent behaviors and local interaction rules. We evaluate it via two application scenarios in order to link together all the components of the system. Comparison to related work showed that our approach is one of the first fully-distributed embedded platforms that makes use of the *Spatial Computing* paradigm for fast-prototyping of interactive design installations. As future work, we identified several directions. We will run large-scale experiments by making use of the entire size of the *ProtoDeck* floor. Secondly, the GUI will hide some of the complexity related to expressing agent-level behaviors and will contain more complex aggregate primitives. Complete design ideas will be prototyped and tested in order to improve our methodology based on user feedback.

8. REFERENCES

[1] H. Abelson et al. Amorphous computing. *Communications of the ACM*, 43(5):74–82, 2000.

[2] T. Baumgartner, S. Fekete, T. Kamphans, A. Kröller, and M. Pagel. Hallway monitoring: Distributed data processing with wireless sensor networks. In *REALWSN*. 2010.

[3] J. Beal and J. Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *Intelligent Systems, IEEE*, 21(2):10 – 19, march-april 2006.

[4] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, chapter Organizing the Aggregate: Languages for Spatial Computing. IGI Global, 2012.

[5] T. Bekker, J. Sturm, and B. Eggen. Designing playful interactions for social interaction and physical play. *Personal and Ubiquitous Computing*, 14(5):385–396, 2010.

[6] N. Biloria. Emergent technologies and design. *eCAADe 23*, pages 441–447, 2005.

[7] A. Crabtree, T. Hemmings, and T. Rodden. Pattern-based support for interactive design in domestic settings. In *DIS 2002 Proceedings*, pages 265–276. ACM, 2002.

[8] T. Delbrück, A. M. Whatley, R. Douglas, K. Eng, K. Hepp, and P. F. Verschure. A tactile luminous floor for an interactive autonomous space. *Robotics and Autonomous Systems*, 55(6):433–443, 2007.

[9] S. Dulman. *Robotics in Architecture*, chapter Practical Programming of Large-Scale Adaptive Systems. JapSam Books, 2012.

[10] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *DCOSS*, volume 3560 of *LNCS*, pages 466–466. 2005.

[11] M. Haeusler. *Media facades: history, technology, content*. Avedition, 2009.

[12] J. Hubers. Collaborative design in protospace 3.0. *Changing roles; new roles, new challenges*, 2009.

[13] B. Knep. http://www.blep.com/healingPool/.

[14] J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wirel. Netw.*, 8(2/3):169–185, Mar. 2002.

[15] N. Lehrer and S. Rajko. Thrii. 2010.

[16] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Trans. Softw. Eng. Methodol.*, 18:15:1–15:56, '09.

[17] R. Nagpal. *Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics*. PhD thesis, Massachusetts Institute of Technology, 2002.

[18] B. Quinn. *Textile Futures: Fashion, Design and Technology*. Berg Pub Ltd, 2010.

[19] U. Wilensky. Netlogo, 1999. http://ccl.northwestern.edu/netlogo/.