Applying Spatial Computing to Everyday Interactive Designs

Stefan Dulman
Centrum Wiskunde & Informatica
The Netherlands
Email: dulman@cwi.nl

Chris Kievid
Chris Kievid Design
The Netherlands
Email: info@chriskievid.nl

***Abstract*—In this position paper, we address the applicability of spatial computing in the field of interactive architecture. The process of designing large-scale interactive systems is cumbersome, due in fact to single design cycles (transforming ideas into prototypes) taking a period of time usually measured in months, most of it dedicated to writing the software controlling the system. As most interactive architecture projects pass through several design cycles interleaved with user studies, speeding up the generation of the needed software becomes of crucial importance. The global-to-local programming approach is in fact a perfect tool for this task. Describing complex behaviors with simple rules is rarely seen in the existing installations, the large majority employing a central computer for the control of the system. Building up on our previous experience in this area, we created HiveKit, a proof of concept allowing bridging between the two fields, giving non-specialists easy access to distributed algorithms. HiveKit is a software package which allows designers to specify the desired behavior and automatically generate and deploy the needed code on networks of embedded devices. We introduce several projects where HiveKit is employed and create an argument, based on user studies, favoring the need for large-scale adoption of such tools.**

I. Introduction

The built environment in which we live is in a continuous evolution. Smart buildings and smart infrastructure, part of the biggest vision of smart cities [19] are already around us, with their numbers and capabilities increasing with advances in technologies such as internet-of-things. The shifting patterns of human interaction within the contemporary social and spatial reality intensifies the demand for integrating networked embedded systems in the physical environment [1]. We advocate that emergent behaviors in such distributed spatial systems is a desirable feature for designers and should be promoted rather than prevented. They promise creation of unprecedented spatial qualities and experiences, and have the potential to perform as catalysts for numerous social processes and transformations, such as dynamic demands and values and to different patterns of use [4] [8] [12].

However, the fact that computation has become omnipresent, has only shown that the real question that we are now facing is: how can we take real advantage of its potential? With the number of "smart" devices surrounding us continuously growing, traditional interaction modalities based on direct control by the user of any given device stop to work. How can we interact with ease with an environment filled with many interactive devices? The problems become further complicated when we take into account numerous users that may occupy an interactive environment. In those situations not only interactions between a person and a cloud of devices need to be taken into account, but also interpersonal relations and social interactions, including potential conflicts, need to be addressed and resolved. Furthermore, a significant research question is that of an unknown affordance. Since interactive environments are still uncommon, there are no cultural models or patterns for interacting with them.

While autonomous agency has been advocated by leading design professionals [18] [15] as the preferred path, it can be postulated that actual architectural design and implementations are virtually inexistent [13]. As most systems operate in isolation or even at cross purposes, designers lack technical knowledge required to formulate, explore and validate creative use, and additionally rapid prototyping and experimental testing of resulting embedded networks can be challenging. It becomes evident that a designer-friendly software tool for exploring the interaction user-distributed systems is desirable.

A wide range of software tools and plugins exists to support architectural design or simple interaction prototyping. Despite that, there are virtually no software platforms that would allow designers to easily simulate and explore complex and emergent behaviors in systems built out of many embedded nodes, prototype and eventually deploy those systems. The reasons underlying this state of facts lie in a missing methodology for programming complex systems in manner that exploits stable emergent behaviors while providing the user with the needed control. Designing specific distributed algorithms for a given application is usually preferred to stacking complex behaviors occurring from simple local rules interactions.

Designing the software for a complex large-scale distributed system is a daunting task. From our experience, simple interactive installations containing a few tens of sensors and actuators can keep a software team busy for months [5]. We would like to point out the true limitations of this long design cycle leading to a clash in cultures. On one hand, the software team needs to work with clear requirements towards delivering a working product [3]. On the other hand, designers work by engaging in design cycles involving fast creation of a prototype, engaging in user studies and integrating the feedback in the next prototype. For every product, a few tens of iterations are common. From this perspective, it becomes clear why the vision of interactive environments is so far away from realization. The prohibitively large duration of a single design cycle (measured in months or even years) prevents the large majority of designers from experimenting in the field of

large-scale interactive systems, let alone exploring the usage of local interactions in their work [17].

In this paper, we will focus on the problem of reducing the duration of design cycles, achieved by integrating theories and results specific to spatial computing into CAD software tools [6]. Our efforts led to the creation of a software tool entitled HiveKit. In the following, we will briefly introduce HiveKit and analyze how employing it led to the development of a number of prototypes, not feasible otherwise under the resource and time budgets allocated to them. We will briefly describe the technical characteristics - the focus being, on purpose, on the benefits and the novel avenues opened by the increased productivity of using spatial computing in interactive architecture.

II. HiveKit

In this section we introduce HiveKit [11], a software tool allowing designers and architects to design behaviors and program distributed networks of embedded devices directly from a CAD tool. Before going into details, we would like to point out a few software tools already in use by the designer community.

Architects traditionally used CAD tools such as AutoCAD in their projects. Recent years have seen a tremendous increase in popularity of the Rhino3D software package [21], one of the reasons being the availability of the Grasshopper plugin [10], allowing parametrization of the objects described in the graphical user interface. Grasshopper bridged the gap with the actual IT technology, allowing the shape of an object and its functionality given by embedding electronics to be described and explored in the same simulator. For example, the Firefly plugin [7] allows automatic code generation for a single Arduino device embedded in the designed object. In the same line, the gHowl plugin [9] allows code generation for a smart phone used in the design. These capabilities are very limited when comparing to a tool such as Matlab/Simulink which allows the generation and deployment of complex control code on devices. Unfortunately, Matlab is highly sophisticated, requiring a proper training in engineering and software technology - it is therefore out of reach for the designer community.

Processing [20] is another tool used to explore complex systems. The behavior of the objects is described in a language similar to C/C++. Despite the impressive graphical results obtained by using this tool, the users need a deep understanding of programming languages and there is no code generation for real devices included. At the other side of the spectrum lies MaxMSP [16], the tool of choice for controlling interactive installations such as light and sound systems using standard communication interfaces, e.g., DMX controllers. The output of the behavior described in MaxMSP can be translated into the proper electrical signals to control a predefined setup of elements. Unfortunately, none of these tools offers a method for controlling a network of devices as a whole, the ones who allow interfacing with hardware elements focusing on the control of each individual element at a time.

As shown below, HiveKit hides from the user all aspects linked to low level virtual machine details, communication protocols, the implementation of distributed algorithms, even the compilation and deployment of code. The designer has only to focus on the behaviors she wants to showcase in the interactive installation. HiveKit consists of two major software components: the embedded systems interpreter and a high-level graphical user interface (GUI), characterized briefly below.

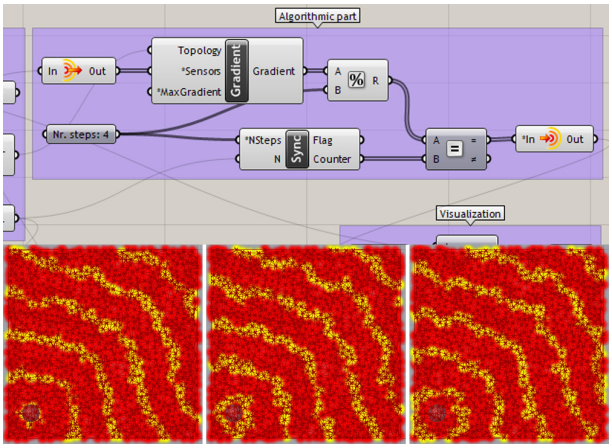

Fig. 1. HiveKit example - top part shows the behavior description; bottom part shows the results of three consecutive executions steps.

A. Embedded software interpreter

The embedded software interpreter allows the execution of behaviors specified in the GUI on Arduino Due devices [2]. It takes as input a desired behavior specified in a compressed XML format. Each component in the GUI (algorithms, mathematical operators, etc.) has its equivalent in a software module which acts as an independent instruction.

HiveKit is designed around the concept of execution rounds. Similarly to the abstract definition of execution rounds, during each round, each node collects information produced locally or in the neighborhood in the previous round and performs a single computation on this data. The start of a round is signaled by an internal clock, which keeps track of the round number. Round sizes can be varied on the present hardware platform from 50ms to a few seconds - the default value being 250ms. If desired, the execution can be synchronized in the network in a distributed manner. By using the firefly synchronization algorithm [23] (available via the Sync module), all the nodes in the network synchronize their execution rounds with their neighbors.

There are two types of modules: *functions* and *algorithms*. *Functions* are pieces of code which can be executed immediately, with the local data available on the node (such as an addition operation applied to a sensor data value and a constant). Functions may maintain state information across different execution time steps. Examples of stateless functions are the arithmetic and logical operators. Examples of state-based functions include the blocks in the digital logic category (such as the Delay block which acts as a buffer for the input data). *Algorithms* are blocks which require data exchanges with the neighbors. The Gradient block is an example - it queries the neighbors for their gradient value and has as output the smallest found one incremented by one. The implementation of algorithms is based on a template that handles data collection from the neighbors, data expiration in the local buffers, error

handling and all the associated operations involving memory management.

The modules execute in the order specified by the compressed XML format. They are linked by data links, called *signals* which are represented internally as float values. Signals flow through all the modules, and each module creates a value at each time step. Signals can be linked to either modules or hardware components such as pins and communication ports. HiveKit provides a few simple drivers for the control of commonly used hardware components such as stepper motors. From the software perspective these drivers are seen as functions, so there is no special handling needed.

B. Graphical user interface

HiveKit functionality can be accessed as a plugin in the Grasshopper software. Grasshopper is a front end for the Rhino3D CAD tool. It is conceptually similar to the Simulink interface in Matlab and uses the same concepts: functions are hidden by blocks connected together in a configuration specified by the user. Each block presents a parameter interface and the user can set global parameters for the simulation as well. Blocks in the basic set offer functionality linked to the creation and complex manipulation of basic geometric primitives. For example, users can rotate a box around an axis or create and modify a mesh surface from a set of initial points. Grasshopper can be easily extended by adding more blocks to this basic set.

Data exchanged between modules can take a variety of forms in Grasshopper. We limited our approach to lists of float numbers, giving the user a global view of the network. The lists have a number of elements equal to the size of the real network - each value in the list corresponds to a value of the signal on a specific network node.

HiveKit adds several blocks to the Grasshopper interface (distributed algorithms, handling of inputs and outputs such as sensors and actuators and various utility blocks such as digital logic blocks). The distributed algorithms set includes at this moment four blocks: Random (activates a specified random fraction of nodes in a network), Sync (triggers the firefly synchronization algorithm), Gradient (a multi-source version of the basic flooding/diffusion gradient algorithm) and Token (a self adaptive algorithm leading to a single token being passed in a linear network).

The second set of blocks targets inputs and outputs. We have added a few pass-through blocks which have no real effect on the simulation but signal HiveKit the interfacing with real sensors and actuators. They are mapped on the capabilities of the hardware platform: sensors include on/off digital sensors and analog-to-digital converters, while actuators include on/off digital actuators, digital-to-analog converters, pulse width modulation outputs and stepper motor outputs. Also in this category we include blocks which allow the user to specify programming loops. By default, Grasshopper prevents data loops from being specified (the solver simply stops with an error when loops are detected). We introduced blocks that allow transport of data signals between any two points in the simulation, bypassing the checks of the solver (similar to the go-to instruction in low level programming languages). The addition of these blocks opens the possibility

Fig. 2. Hive Panels - interactive installation developed by Hive Systems and Chris Kievid Design for the Glow Festival of Light, Eindhoven 2013.

of implementation of any type of digital filter, allowing the user to easily specify common behaviors such as fading in/out of lights, control loops built on sensor/actuator interaction, etc.

The third group of blocks includes blocks specific to digital logic. Grasshopper lacks a proper representation of time - being a CAD tool, the simulations were supposed to be a one-shot simulations, with data passing only once through each block - in other words, the models run once, to completion. We added a number of blocks equivalent to flip-flops and clocks. HiveKit blocks understand the concept of clocks which can be added and simulated at various frequencies. The user is not limited in the number or types of clocks he would like to use.

Figure 1 shows an example of such a behavior (we zoomed in on the relevant part of the simulation). Assume a network of randomly deployed nodes, organized in a mesh topology like the one in the image. We would like to create a wavy pattern, starting at some nodes that get triggered via, for example, a presence sensor which moves away from these nodes in time. The core of the simulation is given by the interplay of two algorithms: Gradient and Sync. In the current setup, Sync allows each node in the network to synchronize its round with its neighbors. The Counter output of this algorithm is the round number, which is the same on all nodes, and ranges in this particular case from 0 to 3. This number increases (modulo 4) each round. The output of the Gradient block is a number reflecting the distance in hops from the current node to the

closest triggered sensor. In this simulation we take this number, perform a modulo 4 operation on it and check if it is equal to the round number. If equal, then an actuator (a yellow light in this case) is triggered. With time, the pattern evolves as given by the output of the sync algorithm, leading to the desired behavior. Each signal in this simulation is a list of values corresponding to the values on each node. The visualization in the simulator is achieved by a number of blocks which will be ignored in the embedded systems code (described in the next section).

The selection of blocks offered by HiveKit will increase with time. Interestingly enough, many complex behaviors can be easily decomposed in these basic components (mapping on the idea that almost any behavior seen in nature can be decomposed in combinations of diffusion, timing and restriction blocks). All the behaviors described in Section III were achieved by using only the presented components.

C. Code generation and deployment

The "brains" of HiveKit are given by the Controller block. This block performs all the needed operations, transforming the desired behavior from the GUI into actual code running on the nodes and also initiates the automatic deployment of code in the network. The goal of this whole project was to allow non-specialists to program large-scale networks by hiding the underlying complexities. Together with the built-in communication and error handling mechanisms, the Controller block allows us to achieve just that. The main automation features provided are as follows:

- **Behavior synthesis** - by right-clicking the Controller block, the user triggers the analysis of the behavior in the GUI. The Controller block creates a list of all blocks and the netlist associated with the simulation. Then, it identifies the blocks relevant to the real-life deployment (ignoring for example the blocks used for visualization - the yellow and red spheres in the previous example). It does that by identifying the active actuators in the simulation and following back the blocks from outputs to inputs until it reaches constant values or sensor inputs. Specific care is taken in the case of loops, timing blocks, blocks with parameters and the execution order of the blocks. Once a valid configuration is created from the GUI, a compressed XML format is generated putting all the blocks, signals and constant values together.
- **Code deployment** - by default, the nodes in the network exchange messages with each other, once every single round. The content of the messages is dictated by the distributed algorithms employed in the current behavior. Each message contains also a version number of the current behavior, version which is incremented with each new deployment. Receiving a message having a different version number than the current one, triggers the node to ignore its contents and, if more recent than its own, initializes a protocol via which the node is requesting the new behavior from its neighbor. Behaviors require more than one message to be transmitted and a transport protocol is put in place to take care of behavior reconstruction from several messages, error handling, retransmission, loss of the source of the message, etc. The final effect is that if a single node (the one connected to the computer) is "infected" with a new behavior, it will spread it in the network virally, without any additional operation required from the user.
- **Logging** - a logging system is put in place and nodes can be triggered to offer runtime execution information on demand. Each block offers several levels of reporting, all customizable by the user (the same holds for all major software components of the embedded software interpreter itself). This way, understanding what goes on inside a node at runtime is an easy operation, starting and stopping different levels of logging at runtime being feasible without the need of restarting or reprogramming the network.

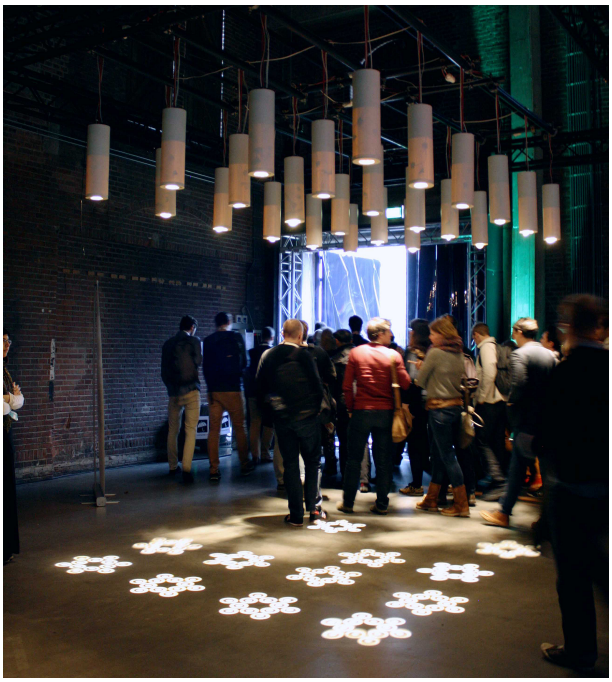

Fig. 3. The Seina project - developed in the "Interactive Environments" Minor at the Delft University of Technology. Instructors: A. van der Helm, M. Rozendaal, C. Kievid.

D. Brief technical details

The embedded system interpreter is developed mainly in the C language as a single thread software component, easy to integrate with other pieces of embedded systems code such as specific systems software. The current prototype targets Arduino Due boards, powered by an Atmel SAM3X8E ARM Cortex-M3 CPU. The flash image (with the full logging capabilities turned on and all the algorithmic blocks enabled) is around 60kBytes of compiled code. The amount of RAM used by the interpreter depends on the size of the programmed behavior, with typical values less than 10kBytes. Memory management is simplified to the maximum - the only memory allocation/deallocation operations are performed when a new behavior is downloaded into a node, apart from that memory management is not in use.

We are making use of all five serial interfaces of the Arduino Due boards (1 UART and 4 USARTs) for creating networks. Each device in the network can be connected thus to maximum five neighbors via wired links. The embedded systems interpreter is agnostic to the communication means

- as long as data is supplied to it in the proper format. This allows easy extension of the communication to the other communication interfaces present on the hardware platform and extensions to radio communication interfaces such as XBee.

The GUI is based on the .NET framework (the large majority of our custom components being coded in VB.Net). Our code makes use of the APIs provided by Grasshopper and Rhino3D. We built most of the custom components around a component template adding clock information and specific parameter handling to the software blocks. As described above, HiveKit allows easy integration of a large range of hardware devices - enabling the use of many sensors/actuators shields already developed for the Arduino platform.

III. Deployment examples

In this section we introduce three example of deployments, showcasing the possibilities of HiveKit as a design platform. Although quite different in their aims and accomplishments, these projects employ a distributed systems approach with meaningful interaction use-cases and user generated content. All projects display meaningful interaction use-cases where the aggregate spatial properties become translated to their corresponding local actions. The behaviors include sensing, actuation and dissemination of information within localized neighborhoods.

The first application that demonstrates the potential to create a precedent for an entirely new class of architectural and related design interventions was protoDECK [14]. The purpose of this prototype is exploring the conceptualization and implementation of ubiquitous deployment of an intelligent visual surface in a spatial environment and enabling a variety of novel applications. This floor, comprised of 168 interrelated tiles, each equipped with multicolor light and the ability to detect presence of persons, allowed invited artists to investigate the various spatial behaviors that such a system is capable of. It served as a first prototype, allowing us to explore and converge the features of HiveKit. It also facilitated the interaction with a large number of students in industrial design and interactive architecture, shaping the interface and functionality of the final prototype.

Bringing digitally-enhanced social interactions to the public domain and creating new forms of interactions wherein architectural components play a leading role is further explored in a second project. The objective of this prototype is to assess the effectiveness of the individual involvement and validate the public awareness, understanding and appreciation on how distributed technologies can serve as platforms encouraging creative public behavior. The Hive Panels installation (Figure 2) consists of eight autonomously operating elements, which can be magnetically attached to any metal surface, in any configuration. Each element is equipped with a powerful multi-color LED array, which can dynamically change its orientation. All elements are capable of sensing presence and motion of persons or objects in front of them using infrared sensors. The Hive Panels can be synchronized to perform complex global-scale tasks in unison, while being triggered only by local stimuli - for example, an elaborate light and motion show that dynamically reacts to passers-by.

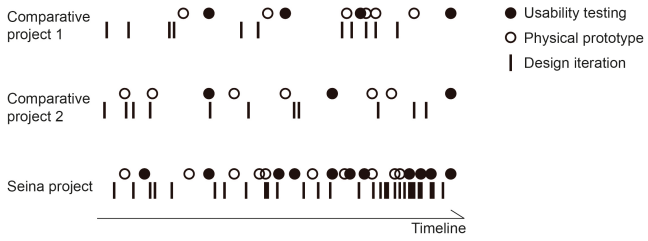

Fig. 4. Comparison between parallel ongoing student projects in the "Interactive Environments" Minor at Delft University of Technology. The horizontal timeline spans between September 2013 and February 2014.

Furthermore, the installation can also exhibit a higher-order behavior and offer practical functionality such as directing light to areas where people are present and adjusting its colors and intensities to fit those people current activity.

The interactive light installation Seina [22] (Figure 3), designed by undergraduate design students, features the ability to extend the developed systems functionalities beyond the initial set of requirements. Due to its flexible system architecture non-experts are be able to design and integrate distinct functionalities after the deployment has taken place. These functionalities include both sensing and actuation modalities but also additional, more immersive and meaningful interactive behaviors. In the current state of development, this prototype consists of 24 autonomously operating lamps suspending from the ceiling. Participants engaging with the installation trigger dynamically altering light patterns on the floor. Demonstrated behaviors include light trails behind moving users, light paths that guide them to other participants or locations, and a variety of shared interactive games and experiences for bringing together and socially engaging various user groups.

IV. Discussion and conclusions

It is difficult to precisely evaluate the shortening of design cycles due to the usage of HiveKit. Nevertheless, we make use of data collected in the Interactive Environments Minor program at Delft University of Technology. Three projects were developed in parallel by three groups of students, similar in complexity and making use of similar hardware. Seina was one of the projects, and the only one in which the students used HiveKit. Figure 4 plots the design cycles along a timeline. It is worth noticing that the usage of HiveKit allowed the students to build roughly double the number of prototypes when compared to the other two projects. The easiness in changing behaviors and deploying them on the fly led also to a significantly increased number of usability tests and design iterations. The final results also reflected this, Seina being significantly more mature than the other designs. The results of surveys, expert panels and direct observations during the various project, let us draw a few conclusions and guidelines for similar systems:

- An intuitive graphical user interface allows designers to visualize faster the emergent properties of their designs with respect to human interaction and the technology they are immersed into. Due to the almost instant viral deployment of new behaviors in the network, the user studies can cover a lot more alternatives.

- The prototyping and deployment activities are significantly shortened by the abstraction from technical concepts such as: real-time event handling sensor information dissemination, precise actuation control, communication protocols, error handling mechanisms, etc.
- The user interface allows full customization of the sensor and actuator capabilities, removing the need for manual implementation of the embedded systems software. Especially for non-specialists this is a very important step giving them access to an otherwise non-usable technology (for example, none of the students in the Seina project had a background in computer science, embedded systems or any related discipline).
- HiveKit leads to a tremendous acceleration for the exploratory phase of the design process as there is no need to involve specialists in embedded systems, distributed algorithms or sensor technology.
- Designers rapidly explore a large number of interaction patterns and focus more on the ergonomic aspects rather than technical ones. Designers perform more early stage user evaluation of proposed interaction scenarios using experiential prototypes.

The approach taken by HiveKit opens quite a number of interesting possibilities of cooperation between the specialists in the fields of distributed systems and large-scale interactive systems design. By hiding functionality in simple blocks and taking the effort of adding automatic reconfiguration options, one enables designers to use complicated mathematical tools in a very intuitive way. Think for example, of the "simple" problem of changing the intensity of lights emitted by light poles on a street such that the amount of street lighting is always constant, despite atmospheric conditions, broken lamps, etc. A specialist would recognize this problem belonging to the class of distributed constraint optimization problems (DCOP) and might be able to suggest a few proper algorithms. Needless to say, the non-specialist has little idea that such mathematical tools exist, let aside experiment with their use in real scenarios. This motivates us to add a block for DCOP in HiveKit in the near future.

Another interesting direction for development is the integration of smart phones with the design tools. People interacting with the installations might do so via their smart phones - generating automatically the code for an app and testing its effects directly in a CAD tool would reduce tremendously the development cycle. An initial step has been taken in the form of the gHowl plugin [9], which shows how to integrate a single smartphone. Future work with respect to HiveKit will address also an extension from this perspective.

The Seina and Hive Panels interactive projects demonstrate that a fusion between the material, the electronic and the digital domains is now within reach of non-specialists. Compared to existing approaches, our proposed solution of automating code generation for the network of embedded systems allows vast new possibilities for creation of new forms of interactions. At the same time, it reduces the needed infrastructure and increases scalability, robustness, availability and easy deployment and ultimately reduces the cost of interactive installations. In the words of Mike Kuniavsky: "Right now is the time to create a practice of ubiquitous computing user experience design. The technology is ready. Consumers are ready. Manufacturers are ready. The world is ready. Now it's up to designers to define what that practice will mean." [15]

ACKNOWLEDGMENTS

The authors would like to thank Tomasz Jaskiewicz and Andrei Pruteanu who were part of the Hive Systems initiative and took active roles in the development of HiveKit.

REFERENCES

[1] Alive2013: International Symposium on Adaptive Architecture. [Online]. Available: http://alive2013.wordpress.com/

[2] Arduino Due. [Online]. Available: http://arduino.cc/en/Guide/ArduinoDue

[3] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*. IGI Global, 2012, ch. Organizing the Aggregate: Languages for Spatial Computing, pp. 436–501.

[4] Building Dynamics: International Symposium on Exploring Architecture of Change. [Online]. Available: http://buildingdynamics.org/

[5] S. Dulman, "Data-centric architecture for wireless sensor networks," Ph.D. dissertation, University of Twente, The Netherlands, 2005.

[6] S. Dulman, "Practical programming of large-scale adaptive systems," in *IA5, Robotics in Architecture*, K. Ooosterhuis and H. Bier, Eds. Jap Sam Books, 2012.

[7] Firefly. [Online]. Available: http://fireflyexperiments.com/

[8] M. Fox and M. Kemp, *Interactive Architecture*, 1st ed. New York: Princeton Architectural Press, 2009.

[9] gHowl. [Online]. Available: http://www.grasshopper3d.com/group/ghowl

[10] Grasshopper. [Online]. Available: http://www.grasshopper3d.com/

[11] HiveKit. [Online]. Available: http://hive-systems.net/hivekit, developed by Stefan Dulman, Andrei Pruteanu and Tomasz Jaskiewicz

[12] M. Hosale and C. Kievid, "Modulating territories, penetrating boundaries," *Footprint: Delft School of Design journal*, vol. 6, pp. 55–67, March 2010.

[13] T. J. Jaskiewicz, "Towards a methodology for complex adaptive interactive architecture," Ph.D. dissertation, Delft University of Technology, 2013. [Online]. Available: http://dx.doi.org/10.4233/uuid:a81827c5-7d65-4cc7-9fab-20fab3a14c30

[14] S. Karger, A. Di Figlia, M. Bos, A. Pruteanu, and S. Dulman, "Spatial computing for non-it specialists," in *Spatial Computing Workshop (SCW2012), AAMAS2012*, Valencia, Spain, 2012.

[15] M. Kuniavsky, *Smart Things: Ubiquitous Computing Experience Design*, 1st ed. New York: Morgan Kaufmann, 2010.

[16] MaxMSP. [Online]. Available: http://cycling74.com/products/max/

[17] A. Nereim, "Emergent Behavior in Networked Architecture," in *ACSA Teachers Conference Proceedings*, 2008.

[18] K. Oosterhuis, "Swarm Architecture II," in *The architecture Co-Laboratory: GameSetandMatch II*, K. Oosterhuis and L. Feireiss, Eds. Delft, The Netherlands: Episode Publisher, 2006, pp. 14–28.

[19] K. Oosterhuis, *Hyperbody: First Decade of Interactive Architecture*, 1st ed. Ram Publications, 2012.

[20] Processing. [Online]. Available: http://processing.org/

[21] Rhino3D. [Online]. Available: http://www.rhino3d.com/

[22] Seina, http://seina.nl/ - developed in the "Interactive Environments" Minor program at the Delft University of Technology. Instructors: Aadjan van der Helm, Marco Rozendaal, Chris Kievid. Students: Tom Hemmes, Assmae El Coudi Amrani, Tomas Giele, Jules Dudok, Teresa Maria Martin de la Sierra Baena, Dima Politin, Thijs Langbroek, Jesse Beem, Donna Stam.

[23] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 142–153. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098934