An Educational Tool for Interactive Parallel and Distributed Processing

Luigi Pagliarini[1,2] Henrik Hautop Lund[1]

[1] Centre for Playware, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[2] Academy of Fine Arts of Bari, Via Gobetti, 8. 70125 Bari, Italy

luigi@artificialia.com hhl@playware.dtu.dk
www.playware.dk

Abstract

In this paper we try to describe how the Modular Interactive Tiles System (MITS) can be a valuable tool for introducing students to interactive parallel and distributed processing programming. This is done by providing an educational hands-on tool that allows a change of representation of the abstract problems related to designing interactive parallel and distributed systems. Indeed, MITS seems to bring a series of goals into the education, such as parallel programming, distributedness, communication protocols, master dependency, software behavioral models, adaptive interactivity, feedback, connectivity, topology, island modeling, user and multi-user interaction, which can hardly be found in other tools. Finally, we introduce the system of modular interactive tiles as a tool for easy, fast, and flexible hands-on exploration of these issues, and through examples show how to implement interactive parallel and distributed processing with different software behavioural models such as open loop, randomness based, rule based, user interaction based, AI and ALife based software.

Introduction

Parallel and distributed processing has been an important subject within computer science and artificial intelligence for decades, and is one of the major focus points in most computer science curricula and theoretical educational textbooks. It is normally viewed as an important subject to teach computer science and engineering students, since numerous applications and systems are based on the principle of parallel and distributed processing, including the Internet, cloud computing, parallel computers, multi-agent systems, swarm intelligence, etc. There are numerous important issues related to parallel and distributed processing that a student has to learn about. Within *algorithmics*, it is important to learn to what extend parallelism can improve efficiency and what kind of algorithms can exploit parallelism. This leads, for instance, to a demand for knowing about hierarchical and functional decomposition of problems. An educational tool for this kind of algorithmics learning should allow students to learn about when to utilise shared variables (e.g. in the master) and distributed variables, when to use a scheduler (in the master), how to use semaphores for critical sections, and for instance allow students to confront the mutual exclusive problem [1]. Also, general computer science learning about *operating systems* demands learning about distributed systems, and the issues related to topology, communication, event based control, prevention of deadlocks, data transfer, etc. (e.g. [2]). Obviously, learning about *artificial intelligence* also demands learning about distributed systems for learning about artificial neural networks, evolutionary computation, multi-agent systems, swarm intelligence, etc., including also learning of *artificial life* and *robotics* (e.g. multi-robot systems).

A number of these computer science themes can appear quite abstract to the engineering and computer science student. There is clearly a need to have an educational tool that allows the students to confront these themes in a very concrete manner. We suggest that the best way to learn about these abstract issues is through direct *hands-on problem solving*, following the pedagogical principles of Piaget [3] known as constructionism [4, 5, 6] and guided constructionism in the computer science literature [7]. We combine this with an approach of trying to contextualise IT training for students by allowing them to work with building blocks [8]. Numerous experiments have shown that the hands-on, problem-solving, constructionism approach allow the learner to confront abstract, cognitive problem solving in a simpler manner through the physical representation. The feature that different representations (e.g. physical representation) can cause dramatically different cognitive behaviour is referred to as "representational determinism" [9]. In fact, Zhang and Norman [10] propose a theoretical framework in which internal representations and external representations form a "distributed representational space" that represents the abstract structures and properties of the task in "abstract task space" (p. 90). They developed this framework to support rigorous and

formal analysis of distributed cognitive tasks and to assist their investigations of "representational effects [in which] different isomorphic representations of a common formal structure can cause dramatically different cognitive behaviours" (p. 88). "External representation are defined as the knowledge of the structure in the environment, as physical symbols, objects, or dimensions (e.g., written symbols, beads of abacuses, dimensions of a graph, etc.), and as external rules, constraints, or relations embedded in physical configurations (e.g., spatial relations of written digits, visual and spatial layout of diagrams, physical constraints in abacuses, etc.)" (p. 180) [9].

For the distributed processing education, we suggest using *interactive* parallel and distributed processing that allows the student to easily represent, interact with and create their own parallel and distributed processing system in a physical manner. Here, we will divide the work into some of the sub-problems that the students will have to confront and learn about through practical implementations. These sub-problems include distributedness, master dependency, software behavioural models, adaptive interactivity, feedback, connectivity, topology, island modeling, and user interaction.

Indeed, designing software for *interactive parallel and distributed systems* means moving away from the traditional routes and to face another way of developing algorithms. This other programming paradigm demands the programmer to get into a new "state of mind", which is a most difficult thing to do. It is therefore important to have a clear idea of the concepts and definitions underlying this paradigm of interactive parallel and distributed processing:

Interactivity

For interactivity, here we intend a physical and tangible interaction. The physical parallel and distributed system enables the experience of physically manipulating objects and the material representations of information. The technology embeds physical, conceptual and cultural constraints. The mapping between the physical affordances of the objects with the digital components (different kinds of output and feedback) is a design and technological challenge, since the physical properties of the objects serve as both representations and controls for their digital counterparts [11]. Here, we make the digital information directly manipulatable, perceptible and accessible through our senses by physically embodying it.

While playing with the system, the user can take advantage of the distinct perceptual qualities of the system and this makes the interaction tangible, lightweight, natural and engaging. Interacting with a physical parallel and distributed system may mean jumping over, pushing, assembling, touching physical objects and experiment a dialogue with the system in a very direct and non-mediated way, and hence it is viewed as highly suitable e.g. for student training. Undeniably, this allows for a direct hands-on experience and learning.

Parallel and Distributed

A computational process is called distributed [12] when a single computational atom is on one side autonomous and on the other insufficient to determine the desired outcome. Therefore a computational process will be called distributed when two or more computers – communicating through any possible network - will contribute to accomplish the very same task by sharing different roles in a computational problem or process.

Besides that, whenever considering a *distributed (computational) process*, it is necessary to define the level of parallel vs. serial computational flow that the system should perform, as well as to define the "computational group" characteristics. The Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Since the modular interactive tiles system is mostly dedicated to the task parallelism problem it tends to run distributed processes in, at least, three different ways: *Fully-Distributed, Semi-Distributed* or *Centralized*.

Modular Interactive Tiles System

Under an educational point of view what is really needed, as well as would be a real additional value, is a tool that allows for investigating and understanding parallel and distributed processing meanwhile stressing the user and/or multi-user interactivity component. One possibility is the Modular Interactive Tiles System (MITS) may provide novel programmers with such a tool and approach, since the system is based on robotic modules with certain properties: Each robotic module has a physical expression and is able to process and communicate with its surrounding environment. The communication with the surrounding environment is through communication to neighbouring robotic modules and/or through sensing or actuation. A modular robot is constructed from many robotic modules.

The MITS approach inherits the behaviour-based robotics methods [13] and exploits it with the belief that behaviour-based systems can include not only the coordination of primitive behaviours in terms of control units, but also include coordination of primitive behaviours in terms of physical control units. We,

therefore, imagine a physical module being a primitive behaviour. Thereby, the physical organisation of primitive behaviours will (together with the interaction with the environment) decide the overall behaviour of the system. Hence, in a similar way to the control of robot behaviours by the coordination of primitive behaviours, we can imagine the overall behaviour of a robotic artefact to emerge from the coordination of a number of physical robotic modules that each represents a primitive behaviour, eventually opened to single/multi user-interaction.

The modular interactive tiles can attach to each other to form the overall system. The tiles are designed to be flexible and in a motivating way to provide immediate feedback based on the users' physical interaction, since following design principles for modular playware [14].

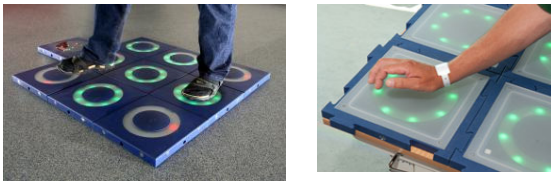

Fig. 1. Modular tiles used for feet or hands interaction.

Each modular interactive tile has a quadratic shape measuring 300mm*300mm*33mm – see Fig. 1. It is moulded in polyurethane. In the center, there is a quadratic dent of width 200mm which has a raised circular platform of diameter 63mm in the centre. The dent can contain the printed circuit board (PCB) and the electronic components mounted on the PCB, including an ATmega 1280 as the main processor in each tile. At the center of each of the four sides of the quadratic shape, there is a small tube of 16mm diameter through which infra-red (IR) signals can be emitted and received (from neighboring tiles). On the back of a tile there are four small magnets. The magnets on the back provide opportunity for a tile to be mounted on a magnetic surface (e.g. wall). Each side of a tile is made as a jigsaw puzzle pattern to provide opportunities for the tiles to attach to each other. The jigsaw puzzle pattern ensure that when two tiles are put together they will become aligned, which is important for ensuring that the tubes on the two tiles for IR communication are aligned. On one side of the tile, there is also a small hole for a charging plug (used for connecting a battery charger), including an on/off switch.

There is a small groove on the top of the wall of the quadratic dent, so a cover can be mounted on top of the dent. The cover is made from two transparent satinice plates on top of each other, with a sticker in between as visual cover for the PCB.

A force sensitive resistor (FSR) is mounted as a sensor on the center of the raised platform underneath the cover. This allows analogue measurement on the force exerted on the top of the cover.

On the PCB, a 2 axis accelerometer (5G) is mounted, e.g. to detect horizontal or vertical placement of the tile. Eight RGB light emitting diodes (LED SMD 1206) are mounted with equal spacing in between each other on a circle on the PCB, so they can light up underneath the transparent satinice circle.

Fig. 2. PCB and components of a modular interactive tile.

The modular interactive tiles are individually battery powered and rechargeable. There is a Li-Io polymer battery (rechargeable battery) on top of the PCB. A fully charged modular interactive tile can run continuously for approximately 30 hours and takes 3 hours to recharge. The battery status of each of the individual tiles can be seen when switching on each tile and is indicated by white lights. When all eight lights appear the battery is fully charged and when only one white light is lit, the tile needs to be recharged. This is done by turning of the tiles and plugging the intelligent charger into the DC plug next to the on/off switch to recharge each tile.

On the PCB, there are connectors to mount an XBee radio communication add-on PCB, including the MaxStream XBee radio communication chip. Hence, there are two types of tiles, those with a radio communication chip (*master tiles*) and those without (*slave tiles*). The master tile may communicate with the game selector box and initiates the games on the built platform. Every platform has to have at least one master tile if communication is needed e.g. to game selector box or a PC.

Fig. 3. Assembly of the modular interactive tiles as a simple jigsaw puzzle.

With these specifications, a system composed of modular interactive tiles is a fully distributed system, where each tile contain processing (ATmega 1280), own energy source (Li-Io polymer battery), sensors (FSR sensor and 2-axis accelerometer), effectors (8 colour LEDs), and communication (IR transceivers, and possibly XBee radio chip). In this respect, each tile is self-contained and can run autonomously. The overall behavior of the system composed of such individual tiles is however a result of the assembly and coordination of all the tiles.
The modular interactive tiles can easily be set up on the floor or wall within one minute. The modular interactive tiles can simply attach to each other as a jigsaw puzzle, and there are no wires. The modular interactive tiles can register whether they are placed horizontally or vertically, and by themselves make the software games behave accordingly.

Fig. 4. Physical interaction with the modular interactive tiles placed on the ground.

Also, the modular interactive tiles can be put together in groups (i.e.: tiles islands), and the groups of tiles may communicate with each other wireless (radio). For instance, a game may be running distributed on a group of tiles on the floor and a group of tiles on the wall, demanding the user to interact physically with both the floor and the wall.

Theoretical Aspects of Interactive Parallel and Distributed Processing

Interactive parallel and distributed systems programming demands the student programmer to shape specific abilities, and we believe that the MITS can simplify this learning process. We will present a number of the interactive parallel and distributed sub-problems that a student needs to learn about, and we believe MITS provides an open tool for facing all the aspects of programming both low and high level programming or front and back end representation.

Classical parallel and distributed processes subtasks
Coding parallel and distributed processes stress programming and understanding of different levels, such as: physical level (i.e.: bit transmission); data link level (i.e.: packages, transmission errors and recovery); network level (i.e.: addresses and packages destination); transport level (i.e.: messages exchanges between clients and master/s); session level (i.e.: defining and implementing sessions in terms of priorities and process-to-process communication); representation level (i.e.: working on data-format differences); application level (i.e.: the end-user interaction and feedback); and to understand and implement solutions for robustness (i.e.: errors diagnosis and recovery); reconfiguration (i.e.: modules assembling); unreliable communication (i.e.: data loss, duplication and corruption); parallelism and concurrency (i.e.: language non-deterministic side-effects); fixed and expanding parallelism (i.e.: modifying the number of involved processors).
It is also essential when teaching information distribution to work on problems such as system connection (i.e.: total vs. partial connection); token-passing (i.e.: how to share and act on critical information); deadlock prevention (i.e.: wait-die, wound-wait, etc.); memory sharing (i.e.: how to locate the physical memory of the distributed system); topology (i.e.: ordinary and complex topology algorithms, initial vs. run-time topology building, etc.); processes transfer (i.e.: distributing the work-load, speeding up calculation, hardware and software specialization amongst the system modules); centralized vs. hierarchy vs. distributed approaches (i.e.: leaded or unleaded information flow); and run-time adaptation (i.e.: adapting the system re/actions on-the-fly).
Besides all of the above "classical" sub-problems of computer science, our platform forces the educational session to face other aspects that software designers should deal with when learning parallel and distributed processing. Such sub-tasks include local and global connectivity, hardware multifaceted topologies, interactivity and adaptive interactivity, and multimodal feedback.

Connectivity
To materialize a proper interactive parallel and distributed platform, the modular interactive tiles system has to implement both a *local connection* system - through which the hardware cells communicate to the neighbourhood and propagate such information from side to side – and a *global connection* device – through which to connect with neighbour platforms and any external tool.

Hardware Multifaceted Topologies
Since the modular interactive tiles system implies the use of run-time de/attachable modules, the emphasis on hardware/software topology is quite strong and it demands a big effort to comprehend the programming

and dealing with such structures. In our model we were able to identify three specific subtypes of topologies:
1. *Regular*, that is a one-block (i.e. any given group of hardware cells attached in a contiguous way and sharing a single master cell) platform with modules attached in a squared or rectangular shape;
2. *Irregular*, which is a one-block platform, which can be arranged in any desired shape. Nevertheless hardware cells have to be continuous (i.e. the assembling does not reveals discontinuity and there is not any isolated cell or group of cells);
3. *Islands Configurations*, that is a platform made with two or more one-blocks (i.e. as defined above in point 1, and 2). It makes no difference whether master cells communicate amongst each others, through an external device, or do not communicate at all.

Interactivity
Implementing software for modular interactive tiles implies designing, or at least dealing with a quite relevant interactive scenario, since in most cases the use of the software itself relies on the users' physical and continuous action. The software designer will have to deal with completely different requirements accordingly to single-user or multi-user targeted software. Often, the software designer will also have to hypothesize a large variety of behavioural situations, even including situations (according to our personal experience) where a single-user platform will be used by many users, or a multi-user software will be run by a single user.

Adaptive Interactivity
The way we approach interaction in such a modular and distributed model leads beyond the classic idea of human-machine interaction (HMI), and is of fundamental importance since it prospects and applies - under both physical and cognitive circumstances – user adaptation and user adaptivity. First of all, our model being architecturally reconfigurable – eventually run-time reconfigurable – represents by itself the essence of adaptation. In addition, being focused on users' physical action, such a system can be easily tailored to users' activity, either in real time or in the long run. To reach such a goal, modular interactive tiles can be programmed using many different strategies that also depend on the quality and quantity of feedback the software designer is willing to exchange with the users. (Feedback and multimodal feedback will be introduced in the next paragraph). Indeed, in more then one case we showed [15, 16] that using modular interactive tiles we could detect some of the users' characteristics, and therefore adapt the software execution to those. Last but not the least, in further tests it has been shown that by capturing the users' provisory attitude and adapting the software execution to that it is possible, in some cases, to eventually modify the users' behaviour itself [16].

Multimodal Feedback
When talking about HMI we kind of committed ourselves to the "how you give is more important of what you give" motto. Therefore, in recent years we pushed our research towards software and tools that can both give and get feedbacks from the user(s).
When developing software for modular interactive tiles we constantly try to provide the user with an *immediate feedback* (e.g. LED, experience report) as well a *delayed* or *long term feedback* (e.g. adaptivity, documentation software). For the immediate feedback from modular interactive tiles we use light (LED) configuration or colours. In addition to that, anytime there is a need for a stronger or a more complex or long-run "signal", we interface the modular interactive tiles with external devices in a layered mode, where each layer of feedback can be added/removed freely on top of each other This is what we call Layered Multi-modal Feedback [17]. The external devices we use can be "passive" as vision oriented feedback (e.g. screen, projector, etc.), sound oriented feedback (e.g. loud speakers, buzzers, etc.), or "active" such as computational devices that through an external communication (e.g. radio and internet) run an analysis or link the user action to specific databases.
In conclusion, to manage and teach the many features of parallel and distributed programming we need to run on a system, which is robust, reliable and easily reconfigurable. This is where we believe that the MITS can express a certain degree of efficiency, besides of being ideal in shifting the level of representation from the very abstract representation to an empirical representation. Therefore, in the following paragraph we provide examples, which attempt to show how one can access the above-described aspects in a fast, comprehendible and easily generalizable way.

Implementations Examples

As a first step the teacher/tutor should introduce students to the hardware platform (Figures 2, 3, and 4) and ask the class to implement all the needed protocols for obtaining a robust, efficient and reliable parallel and distributed system. This would require and encourage students to face the basic algorithms and protocols that the subtasks of parallel and distributed systems need (e.g.: physical level, data link level, network level, transport level, session level, representation level, etc.).
Once such a start-up system is obtained (from the students work or from the pre-made system), a second step could be, for example, testing the system by working on problems such as application, robustness,

communication, system connection, token-passing, deadlock prevention, parallelism, reconfiguration, memory sharing, topology, and process transferring.
The MITS model is ideal for implementing all of the above challenges since the hardware components are minimalistic and the distributed system complexity can be developed and tested in a quick and easy manner (Figure 5).

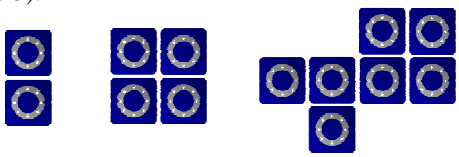

Fig. 5. Examples of different topologies

Once students have reached this new level of competencies, the tutor can drive their attention to a higher level of representation and ask them to implement end-user interaction based applications, such as in the following examples.

Games Examples.
Once a specific topology is chosen, the software engineering student can implement and run a large variety of tasks (here we start by considering examples to apply to a *semi-distributed*, *single user application* on a *regular topology* platform).

Open Loop and *Randomness based* software.
The simplest case, a naïve one, could be the following Easy Game (Figure 6).

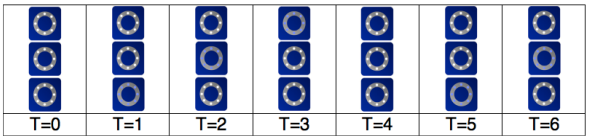

Fig 6. Easy Game, a sequence of 7 states

In Easy Game the light is "passed" from one module to either an adjacent or a distant one (i.e. with a predefined open loop algorithm or randomness based one). In both the above cases the software cycling is endless and we need to introduce the *interactivity level* (e.g. the game finishes when the user hit the lighted tile) to stop it, and by doing so transforming the two into very young children games. When the user press a tile, then the dynamics somehow stops and the tiles freeze in a particular pattern, until the user presses the lighted tile again, and the light shift sequence will start again.

Rule(s) based software.
One step further is a rule-based software characterized by the fact that pattern sequence - which can either be predefined or random based – is governed by a specific rule or set of rules. The simplest case we can think of is the one where, given any machine state and configuration (e.g. two tiles) those states which are ON turns OFF and those states which are OFF turns ON. Of course, we can design a much more complex setting but, essentially, this is the logic that is used in rule based software.
On the other hand, when introducing the interaction element in rule based software we obtain a more dynamic scenario denoted by the fact that the rules and users are coactive and contribute step by step to the system state. Such a situation can be clearly observed in the *American Football* game (Figure 7).

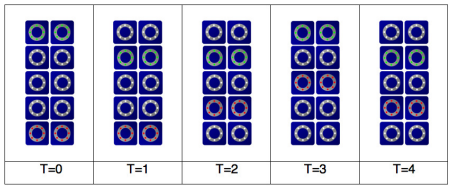

Fig. 7. *American Football*, a sequence of 5 states

This is one-against-one game where, given a, say, 5 (width) per 2 (height) cluster of modular interactive tiles, such interactive software is made so that at the beginning of the game the platform extremes appear activated (i.e. light on) and of two different colours (i.e. blue in one extreme and red in the other extreme). By squeezing the tiles, the user "pushes" the color/activation forward in the row (i.e. switches off the squeezed tile and switches on the adjacent one towards the opponent). The user who first pushes its color to the opposite extreme of the game platform wins the game.

User-interaction based software.
The user-interaction based program is, per se, an interactive software conception in which the user directly contributes to the next machine state (i.e. tiles color or activation). Such a software model is quite similar to the interactive version of the rule based software – since the user itself cannot determine the machine states if not aided by some underlying algorithm. It only differs from that in terms of strain used on increasing the user role and contribution to the next machine state, and the attempt to reduce the rule component. A good example could be the *Final Countdown* game (Figure 8). In the *Final Countdown* the tiles platform can vary both in aspect and size, since the game components behave all in the very same way. It consists of a number of tiles that, when the game is initiated, all of the tiles are fully lighted (i.e. any color would do). After initialization and with a given interval (e.g. one second) they all start to "fade-out" switching OFF one of their 8 light bulbs after the other in a clockwise sequence. If one of them gets completely

OFF the game is over. To restore a single tile to the initial state, the user has to squeeze it. The wider is the platform the more important becomes the strategy users bring into play to keep the game alive.

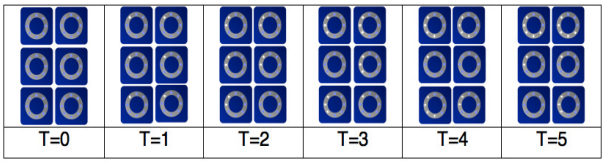

Fig. 8. *Final Countdown*, a sequence of 6 states

A.I. and ALife based software.

The A.I. and ALife based software are, again, a complication of what we defined as rule based systems. Essentially they rely on the same principles, both for the autonomous and the interactive version, although the quality of the computational experience is much higher in terms of software behavioral equality/variety, un/predictability, and etc. Further, since modular interactive tiles tend to resemble pixel-made structures it seems to easily incorporate a consistent number of classical and modern A.I. paradigms. A good example is the Cellular Automata (i.e. CA), a discrete model used in computability theory and many different fields, which consists of a regular grid of cells, each one with a finite number of possible states (e.g. ON, OFF), that can change their state accordingly with their neighborhood activation states [18]. We, first, implemented one of the most famous CA algorithms, the *Conway's Game of Life* on modular interactive tiles and, after that, added the interactive aspect.

Conclusion

We developed the concept of *interactive* parallel and distributed processing in order to put focus on the physical interaction with parallel and distributed system, and to highlight the many challenges that the student programmers might face in understanding and designing interactive parallel and distributed systems.

It is our belief that a system like the modular interactive tiles is a tool for easy, fast, and flexible learning and exploration of these challenges, e.g. as shown with the examples of how to implement interactive parallel and distributed processing with different software behavioral models such as open loop, randomness based, rule based, user interaction based, AI and ALife based software.

Indeed, MITS provides an educational hands-on tool that allows a change of representation of the abstract problems related to designing interactive parallel and distributed systems, so that students can learn about classical and modern aspects of parallel and distributed systems.

Acknowledgements

The authors wish to thank the Centre for Playware colleagues for discussions of the concept and implementations.

References

[1] Harel, D. Algorithmics, Addison-Wesley, 1987

[2] Silberschatz, A., Peterson, J., Galvin, P. Operating System Concepts, Addison-Wesley, 1991

[3] Piaget, J. and Inhelder, B. *La psychologie de L'enfant*. Paris, P.U.F, 1966.

[4] Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.

[5] Papert, S. Constructionism: A New Opportunity for Elementary Science Education. *A proposal to the National Science Foundation*, 1986.

[6] Martin, F. "Ideal and Real Systems: A Study of Notions of Control in Undergraduates Who Design Robots". In Y. Kafai and M. Resnick (Eds.), *Constructionism in Practice: Rethinking the Roles of Technology in Learning*, MIT Press, MA, 1994.

[7] Lund, H. H. "Robot Soccer in Education". Advanced Robotics Journal, 13:8, 737-752, 1999.

[8] Lund, H. H., and Sutinen, E. "Contextualised ICT4D: a Bottom-Up Approach", Proceedings of 10th International Conference on Applied Computer Science, WSEAS, Japan, 2010.

[9] Zhang, J. The Nature of external Representations in Problem Solving. Cognitive Science 21:2, 179-217, 1997.

[10] Zhang, J., Norman, D.A. Representations in Distributed Cognitive Tasks. Cognitive Science 18: 87-122, 1994.

[11] Ishii, H. Ullmer, B. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In Proceedings of CHI: Human Factors in Computing Systems. pp. 234–41, 1997.

[12] Rumelhart, D., McClelland, J., et al., 1986, Parallel Distributed Processing, vol. I, Cambridge, Mass.: MIT Press.

[13] Brooks, R. "A robust layered control system for a mobile robot". IEEE Journal of Robotics and Automation, 2(1):14--23, 1986.

[14] Lund, H. H. and Marti, P. "Designing modular robotic playware," the IEEE Int. Workshop Robots Human Interactive Commun Toyama, Japan. Sep. 27-Oct. 2., IEEE Press, 2009.

[15] Hammer, F. Derakhshan, A., Hammer, F., and Lund, H. H. "Adapting Playgrounds for Children's Play Using Ambient Playware". In Proceedings of IEEE Intelligent Robots and Systems (IROS'06), IEEE Press, Hong Kong, 2006.

[16] Thorsteinsson, T. and Lund, H. H. "Adaptive Modular Playware". (in press).

[17] Lund, H. H. and Thorsteinsson, T. "Social Playware for mediating tele-play interaction over distance" to appear in Proceedings of 16th International Symposium on Artificial Life and Robotics, ISAROB, Japan, Jan. 2011.

[18] Neumann, J. v. "The general and logical theory of automata," in L.A. Jeffress, ed., Cerebral Mechanisms in Behavior – The Hixon Symposium, John Wiley & Sons, New York, 1951, pp. 1-31.