

**nutella: the construction and enactment
of simulated macroworlds**

BY

ALESSANDRO GNOLI

B.S., Politecnico di Milano, 2006

M.S., Politecnico di Milano, 2008

M.S., University of Illinois at Chicago, 2009

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2015

Chicago, Illinois

Defense Committee:

Tom Moher, Chair and advisor

Andrew Johnson

Leilah Lyons

Chris Quintana, University of Michigan

James Slotta, University of Toronto

To my family, for teaching me the hardest thing in life: unconditional love.

ACKNOWLEDGEMENTS

It might sound like a tautology but, for most graduate students, a Ph.D. is an hard endeavor. It certainly was an arduous journey for me and if I have gotten so far I owe it to the patient and constant guidance of my committee and in particular of my advisor, Professor Tom Moher. Thank you so much for your support, your expertise, and for not giving up on me all the times I was ready to quit my Ph.D. and open a gelato shop.

My sincerest gratitude goes also to all the friends at the Learning Technologies Group that shared a piece of (or all) the journey with me and were always there to help and support me.

Finally, I would like to say a big thank you to my family and friends whose love and support carried me through these years in graduate school.

AG

TABLE OF CONTENTS

<u>CHAPTER</u>		<u>PAGE</u>
1.	INTRODUCTION	1
2.	MACROWORLDS	9
2.1	Participatory simulations	9
2.2	Simulated investigations	12
3.	APPLICATION-LEVEL AFFORDANCES AND REQUIREMENTS OF MACROWORLDS	17
3.1	Support multiple macroworld simulations types	18
3.2	Support the leveraging of the physical space of the classroom	20
3.3	Provide feedback during and after the enactment of macroworlds	23
3.4	Support classroom orchestration during the enactment of macroworlds	32
3.5	Support interoperability with other learning technologies	42
3.6	Provide support for “non-functional” capabilities	45
4.	LITERATURE REVIEW	47
4.1	Application frameworks in ubiquitous computing	47
4.2	Supporting the construction and enactment of macroworld applications	56
5.	NUTELLA	63
5.1	nutella’s architecture	63
5.2	framework components / macro-modules	70
5.3	Macroworlds development process with nutella	73
6.	CONSTRUCTING AND ENACTING MACROWORLDS WITH NUTELLA	83
6.1	Support multiple macroworld simulations types	83
6.2	Support leveraging of the physical space of the classroom	85
6.3	Provide feedback during and after the enactment of macroworlds	90
6.4	Support classroom orchestration during the enactment of macroworlds	95

TABLE OF CONTENTS (continued)

<u>CHAPTER</u>	<u>PAGE</u>
6.5	Support interoperability with other learning technologies 104
6.6	Provide support for “non-functional” capabilities 103
7.	EARLY EXPERIENCES WITH NUTELLA 109
7.1	Building RoomQuake with nutella 109
7.2	Building nutella’s macro-modules 112
7.3	Building macroworlds with nutella during a three-day hackathon 114
8.	CONCLUSION, LIMITATIONS AND FUTURE WORK131
	REFERENCES134
	VITA 145

LIST OF FIGURES

<u>FIGURE</u>	<u>PAGE</u>
1. Tablet interface used by the students to collect data about simulated quakes	26
2. Aggregate representation of students' observations in RoomQuake	27
3. Aggregate representation of students' observations in HelioRoom	29
4. Detail of the teacher feedback interface in WallCology	30
5. HelioRoom simulation control dashboard	36
6. RoomQuake administration dashboard	37
7. Demo" and "schedule" mode switch for RoomQuake	39
8. Representational affordances' control in RoomQuake	40
9. Overall architecture of the nutella framework	64
10. Steps involved in the construction and enactment of a macroworld application using nutella	74
11. Output of "nutella new" and "nutella start" commands	75
12. nutella's main interface	76
13. RoomPlaces interface	79
14. Steps involved in the configuration of the classroom space using RoomPlaces (left) and in the creation of RoomCast channels and packages (right)	80
15. RoomCast "classroom remote" teacher interface	97
16. RoomCast channel creation interface (detail)	100

LIST OF FIGURES (continued)

<u>FIGURE</u>		<u>PAGE</u>
17.	RoomCast package creation interface	101
18.	RoomDebugger interface (detail)	105
19.	RoomMonitor interface (details)	106
20.	RoomQuake seismograph interface	110
21.	AquaRoom prototype built by hackathon participants (details)	124

SUMMARY

In the last twenty years the development of new technologies has radically expanded the kind of activity structures that can be designed and built for the classroom. A subset among these technology-enhanced learning environments, that I call *macroworlds*, leverages the notions of ubiquitous computing (Weiser, 1991), distributed interaction (Luyten & Coninx, 2005), and the increasing availability in classrooms of pervasive, non-desktop technologies (e.g. handhelds, large wall-mounted displays, tangibles and many others) to provide engaging ways for students to “experience” and interact with classroom-sized simulations of scientific phenomena. So far, a number of studies demonstrated how *macroworlds* can help students engage in authentic science practices (Duschl et al., 2007), and build meaningful connections between physical activity and important principles in different science domains (e.g. Moher et al., 2010; Enyedy et al., 2012), making *macroworlds* an active area of research both in the field of Human-Computer Interaction and the Learning Sciences (Lindgren & Johnson-Glenberg, 2013).

Despite their promise, *macroworlds* have proven challenging to design, build, and enact, restricting this kind of learning environments to only few exemplars. In particular, one of the main challenges faced by developers while building and enacting *macroworlds* is the lack of a software framework supporting these processes, specifically designed to address the requirements of this learning technology. The goal

SUMMARY (continued)

of this dissertation is to tackle this problem and to devise a method, a set of guidelines and a software framework to support the development and enactment of macroworlds.

In this dissertation, I will first develop a set of requirements that characterize the construction and enactment of macroworlds, based on a review of existing macroworlds, the work of Roschelle and Pea (2002) on Wireless Internet Learning Devices, and especially my experience in dozens of macroworld classroom enactments over the past seven years. I will then introduce nutella, a software framework I created to support the construction and enactment of macroworlds. Finally, I will demonstrate the versatility and design space of nutella by sketching a set of macroworld applications and by describing the experience of a group of developers using the framework during a three day hackathon.

Chapter 1

Introduction

This dissertation introduces *nutella*, a software framework and suite of services, tools, and design templates that support the construction, configuration, and delivery of simulated scientific phenomena to elementary school classrooms. *nutella* is designed to support an emergent genre of learning applications—*macroworlds*—in which location in the classroom is indexed to simulated phenomena made accessible through a networked collection of public and private, embedded and mobile, representational and interactive technologies.

Macroworlds represent an activity abstraction layer designed to subsume two existing genres: "participatory simulations" (Colella, 2000) and "simulated investigations." In participatory simulations, students adopt a first-person perspective, interacting as agents to drive the simulation; the record of their behavior and the emergent results from that behavior become the objects of community inquiry (Colella, 2000). In Colella's virus simulation, students wore small portable computers ("Thinking Tags") around their necks with LED displays showing how many people the person had met during the activity and whether or not they are infected with the virus. At the same time, the thinking tags function as infrared transmitters and receivers giving the students the ability to transfer the virus to others by walking up to them. The goal of the activity was for the students to infer the rules that govern the simulation (e.g., virus

latency, degree of contagiousness, who was the first infected agent, etc.) by recollecting and reflecting on their individual actions and how they affect the emergent outcomes (Chi et al., 2012). In simulated investigations, learners assume a third-person perspective as investigators of an algorithmically simulated phenomena rather than as actors in the simulation itself. For example, in RoomQuake (Moher et al., 2005), learners experienced a series of simulated earthquakes reflected on simulated seismographs positioned at known locations around the classroom. Working in teams, students analyzed the seismograms, from which they were able to determine the distance of the simulated earthquake from each seismograph. Using tape measures anchored at each seismogram, students used trilateration to determine the epicenter of the event, marked on the ceiling of the classroom. As more and more events accumulated, students identified the location of the fault line in their classroom, and identified the temporal pattern and magnitude distribution resulting from the series of quakes.

Like the microworlds (Krajcik & Berg, 1987) that inspire their label, macroworlds offer access to simulated objects of inquiry. Unlike microworlds, in which the user experiences the simulation through the lens of a personal display, in macroworlds users experience the simulation as a kind of sparse immersion through a collection of devices distributed throughout the physical space of the classroom. All students in the classroom engage with the macroworld and with one another in physical space, enacting experiences that serve to generate a data corpus for collaborative knowledge

construction (Scardamalia & Bereiter, 1994; Bielaczyc & Collins, 1999; Slotta & Najafi, 2010) and gain experience with agent behaviors, including science practices (Duschl et al., 2007). Macroworlds seek to leverage the captive audience, tight-knit community, and available space and technologies of classrooms to create opportunities for play (Enyedy et al, 2012), peer interaction (Vygotsky, 1978), autonomy, embodied interaction (Lindgren & Johnson-Glenberg, 2013), and other affordances of the form factor, to support learning and teaching.

Evidence of the effectiveness of macroworlds in supporting learning and teaching is growing, and they have drawn strong interest within the research community. In addition to their “core role” of allowing learners to construct a data corpus (through learners’ activity) to be used in subsequent knowledge construction activities and getting students to enact authentic science practices, macroworlds have demonstrated to help improve students’ attitudes toward science inquiry (Moher, 2008; Moher et al., 2010). The importance of this role play component in science teaching has been highlighted by Jarvis & Pell (2005), which demonstrated how role-play helps student develop an identity as investigators and improve their attitudes toward science. Some macroworld applications take this role-play element a step closer to reality providing students with physical analogs to some of the tools used by scientists, as demonstrated for instance by AquaRoom (Moher et al., 2012). These role play components have also been shown to have an impact on students’ motivation, as do

their desire to play (Enyedy et al., 2012) and their curiosity and motivation to explore. Macroworlds leverage both these elements to maintain kids engaged while playfully exploring and experiencing the simulated scientific phenomena in their classroom. Macroworlds also afford learners to freely move around the classroom allowing them the opportunity to engage in more social exchanges than if they were sitting down in a fixed group. This has two two separate advantages. First, students can exercise more agency in choosing which people they want to interact with. Second, it leverages the richer social interactions among learners as more opportunities to exchange information and learn. Moreover, allowing students to freely roam around the classroom carries three side effects: it fulfills children's need for physical activity and exercise (Strong et al., 2005), it leverages the beneficial role of movement on students' cognitive skills (Sibley et al., 2003) and it allows to make better use of the classroom space allowing kids more "elbow room". Finally, macroworlds leverage both spatial and temporal embodiment to allow students to experience the unfolding of the simulated phenomena in a spatially and temporally situated way. Moving around the classroom and collecting data at different times, students can change their point of view of the phenomena and this can help them better understand the rules of the simulation and create a model of the phenomena (e.g. Moher et al., 2010). The importance of embodiment has strong advocates both in psychology (Johnson, 1987; Clancey, 1997; Clark, 1997; Glenberg,

1997, 1999; Winn, 2003; Lindgren & Johnson-Glenberg, 2013) and human-computer interaction (Dourish, 2004).

In spite of their promise (and a nearly 20-year history), there is only a small collection of applications today that could be considered classroom macroworlds, and none that have been adopted for regular instruction. Two main technological problems limit the growth of the genre. First, there are no specialized frameworks or resources for the development of macroworlds; each of the research groups working in the area maintains separate code bases. This frustrates attempts to share technologies and recruit new members to the development community. Second, there are no standard mechanisms for the delivery of macroworlds, limiting access to classrooms serving as research sites.

nutella addresses both of these problems for a significant class of macroworlds. By providing an architectural framework for macroworld applications, an extensible set of core services (communication, asset tracking, and device management), debugging and monitoring tools, and a library of application and component templates, nutella has the potential to advance the pace and reliable construction of macroworlds. For teachers, nutella offers a web-based configuration interface and run-time controller that allows them to configure and schedule macroworlds for "delivery" to their classrooms through the Internet, for the first time widely creating opportunities for incorporating macroworlds in their instruction. These are novel and potentially impactful capabilities.

We would like to see nutella become the first step toward the growth of a macroworld developer community and "macroworld store" ecology.

The design and development of nutella were guided by two research questions.

Research question 1: What are the application-level affordances that characterize macroworlds and the requirements they impose on technology?

What core capabilities does nutella need to support? The emergent nature of the macroworld genre complicates the design of technologies due to the lack of canonical requirements. To address this problem, I adopted the strategy used by Roschelle & Pea (2002) of identifying a superset of core "application-level affordances" associated with a genre of activity (in their case, mobile computing devices, in mine, macroworlds) through a review of the existing cases of the genre. (In this, I was aided by the fact that I had been involved in the construction and enactment of many of the applications.) These affordances constituted a set of requirements that guided design and whose satisfaction guaranteed that nutella would, at a minimum, enable the construction and delivery of those macroworlds that have been introduced in the literature.

Research question 2: How can we design resources to support the processes of developing and enacting macroworld applications?

I address this question by presenting the capabilities and organization of *nutella*, describing the rationale that led to that design, demonstrating that the design meets the application-level affordance requirements, and describing the early experience of a small group of developers in developing *nutella* applications.

The remainder of the dissertation is organized as follows:

Chapter 2 presents a survey of existing macroworlds applications.

Chapter 3 identifies and formalizes a set of five application-level affordances characteristic of macroworlds and describes the requirements they impose on software infrastructures.

Chapter 4 presents a survey of application frameworks in several ubiquitous-computing research areas that are relevant to this work and reviews research relevant to the five application-level affordances of macroworlds described in the previous chapter.

Chapter 5 describes *nutella*, the software framework I created to support the construction and enactment of macroworlds. The chapter will detail the framework's software architecture, main components and how they interact with each other.

Chapter 6 demonstrates how *nutella* provides support for each of the five application-level affordances identified earlier and how the framework can be used to build macroworlds.

Chapter 7 describes some early experience using the framework. In particular, this chapter will outline the growth of the nutella developers' community over the past year, demonstrate how the framework was used by a group of developers during a three-day hackathon and describe the experiences of the framework contributors.

Chapter 8 presents a summary of the conclusions and contributions of this research, limitations and future directions of work.

Chapter 2

Macroworlds

In Chapter 1, I described a strategy for identifying a core set of macroworld affordances to serve as a requirements base for nutella based on an inventory of reported macroworlds. In this chapter, I provide that inventory, primarily as a reference for the discussion in Chapter 3, in which the set of application-level affordances is developed.

2.1 Participatory simulations

In participatory simulations, students experience and influence the phenomena by taking on the role of an element (e.g., a bee, a car) within a complex system (e.g., a beehive, a traffic jam). Sensor-based devices (e.g., GPS enabled handheld devices) are worn or carried by participants, allowing for the automatic exchange and aggregation of contextually relevant information. Participants experience the simulation both at a local level (i.e., from the perspective of the element they are enacting), and also from a global viewpoint, where they can see how their individual actions affect patterns within the overall system.

2.1.1 Virus simulation

In Colella's virus simulation (2000), students wore small portable computers ("Thinking Tags") around their necks with LED displays showing how many people the person had met during the activity and whether or not they are infected with the virus. At the same time, the thinking tags function as infrared transmitters and receivers giving the students the ability to transfer the virus to others by walking up to them. The goal of the activity was for the students to infer the rules that govern the simulation (e.g., virus latency, degree of contagiousness, who was the first infected agent, etc.) by recollecting and reflecting on their individual actions and how they affect the emergent outcomes (Chi et al., 2012).

2.1.2 BeeSim

BeeSim (Peppler et al., 2010) is a participatory simulation which puts young children in the shoes of honeybees collecting nectar. The simulation makes extensive use of wearable technologies and aims at teaching kids about both the value of communicating nectar sources to other bees and the difficulty of finding nectar. In this context, students assume the role of a honeybee looking for nectar (agent) by wearing a "ForagerBee" glove (a sensor embedded wearable). Kids have only 45 seconds to collect as much nectar as possible while wrestling with the constraints of the system (e.g. limited nectar carrying capacity). Children take turns hunting for nectar and, when a

child's turn is over, they have to pass the glove to one of their teammates. As the glove exchange happens, the child returning from the hunt can try to communicate the location of high-yield flowers to the "next bee", through the use of nonverbal language. Once each child had their chance to wear the glove and hunt for nectar, the team with the most nectar is the "most prepared for winter" and therefore the winning team.

2.1.3 Hunger Games

Hunger Games (Gnoli et al., 2014) is a participatory simulation designed to allow upper elementary school learners to explore fundamental concepts of competitive and cooperative games in the context of animal foraging. In Hunger Games the authors "transform" the physical space of the classroom into a natural habitat containing six food patches of different richness. Each student in the classroom receives a stuffed animal with an RFID tag embodied in it, which acts as his or her "avatar" during the activity. Whenever a student walks to a patch and places their "squirrel" on top of it, the RFID reader embedded in the patch recognizes the tag and starts to provide energy to the students' avatar at a rate dependent on patch quality and competition (i.e. how many avatars are feeding off the same patch at the same time). After the activity students observe and reflect on their individual and collective foraging patterns and design new strategies to improve their individual and/or collective outcome.

2.2 Simulated investigations

In simulated investigations, students assume the role of investigators and scientists tasked with studying and understanding a simulated scientific phenomena co-located within the classroom space. Students monitor and inspect the local state of the phenomena using a collection of media distributed around the room. Over the course of several weeks, students gather and aggregate evidence to answer questions related to the object of inquiry.

2.2.1 Hunting of the Snark

In the Hunting of the Snark (Price et al., 2003), students investigate and discover characteristics of a virtual imaginary creature (called the Snark) hidden within the virtual space, co-located with the physical space of the classroom. Students use a number of physically-digitally coupled tools to locate and interact with the imaginary and elusive creature while it moves across “land, air and water.” Technologies such as handhelds, RFID tags, ultrasound tracking, pressure pads and accelerometers afford students the ability to fly with the Snark, sneak around it while it is sleeping (pressure sensitive floor pads), find its food (handheld and ultrasounds) and take pictures of it using the “Snarkcam” (handheld).

2.2.2 RoomQuake

In RoomQuake (Moher et al., 2005), the classroom is transformed into a seismic area where a series of earthquakes is expected in the next few weeks. Learners experienced the earthquakes through simulated seismographs positioned at known locations around the classroom. Working in teams, students analyzed the seismograms, from which they were able to determine the distance of the simulated earthquake from each seismograph. Using tape measures anchored at each seismograph, students used trilateration to determine the epicenter of the event, marked on the ceiling of the classroom. As more and more events accumulated, students identified the location of the fault line in their classroom, and identified the temporal pattern and magnitude distribution resulting from the series of quakes.

2.2.3 HelioRoom

HelioRoom (Thompson & Moher, 2006) is an embedded phenomena application about astronomy, where students are immersed in a classroom-sized model of the solar system. The sun is hypothetically located at the center of the classroom where four screens, adjacent to each one of the walls, are actually “portals” looking into the virtual space beyond the classroom walls. These portals display eight, equally-sized, colored circles moving at different speeds from the right to the left of the screens and from one screen to the next in counter-clockwise order. The eight circles represent the planets of

the solar system orbiting around the sun. The challenge students have to face is to associate each colored circle with the right planet. During the unit students work with the simulation and collect observations about the planets' speeds and mutual occlusions, trying to accumulate enough evidence to help them associate planets and colors.

2.2.4 WallCology

WallCology (Moher et al., 2008; Cober et al., 2012) is another embedded phenomena application designed to provide elementary and middle school students with a multi-week simulation of population ecology. WallCology situates students as investigators within a complex simulated ecosystem located in the classroom walls. Students can access the state of the simulation through a set of computer screens adjacent to the classroom walls, called "WallScopes" which give them access to distinctive (but connected) local virtual environments containing mold, vegetation and various species of animated creatures crawling over lath and pipes. Students act as ecologists whose goal is to keep the ecosystem (and particularly endangered species) alive. In order to do so, students need to conduct investigations on population estimation the identification and classification of species and focus on topics such as life cycle phases, food chains, predator-prey relationships, habitat selection, response to environmental change and adaptation.

2.2.5 AquaRoom

In AquaRoom (Novellis & Moher, 2011), the latest embedded phenomena application, students take on the role of hydrogeologists with the task of mapping a subterranean aquifer system (mapped to their classroom floor plan) and to use this information to decide where to locate a new chemical plant within the “local community” to minimize potential environmental impacts. In order to accomplish this task, students “inject tracer dyes” and “obtain water samples” using a portable tablet-based “drilling unit.” A suction cup attached to a (non-functional) Ethernet cable is used to select locations for dye injection or water sampling. Test tubes capped with i-Buttons (similar to contact RFID tags) serve as simulated dye sources and sample repositories. Students “inject” dyes by inserting the test tubes into USB readers attached to the tablet drilling unit, with the liquids virtually running through the cabling. The interactive interface on the tablet computer allows them to mark the injection location, based on a grid system defined by the tiles on the room’s drop ceiling. “Water samples” are subsequently collected in a similar fashion, and tested for the presence of dyes using a simulated spectrometer represented by a shared desktop computer with its own USB reader. The injection of a dye followed by sampling allows students to establish the presence of an aquifer and the direction and rate of flow, which are marked on the tablet map and on a collective classroom map.

2.2.6 EvoRoom

EvoRoom (Lui & Slotta, 2013, 2014) is a room-sized, immersive simulation of a rainforest ecosystem modeled after Borneo and Sumatra. The goal of EvoRoom is to teach high-school students about biodiversity and evolution. The simulation unfolds and leverages the what the authors call a “smart classroom”: a room with six projected displays (three on each, opposite side of the room) and two interactive whiteboards in the middle. The six displays on each side of the room display a representation of the Borneo and rainforest. Within this context, students assume the role of “field researchers,” and are tasked with gathering evidence of evolution by comparing simulations from a range of time periods. Working individually and in groups, they observe changes in life forms over time (the room can be “played” through 200 million years of evolution), consolidate their findings as a community, and develop hypotheses about the evolutionary changes that might have taken place.

Chapter 3

Application-level affordances and technology requirements of macroworlds

In an article published in 2002 and titled “A walk on the WILD side, how wireless handheld may change computer-supported collaborative learning” Jeremy Roschelle and Roy Pea discuss the physical affordances of Wireless Internet Learning Devices (WILD) and outline how the use of this technology might reshape activity structures in K-12 classrooms. As part of their argument, the authors survey a set of early WILD applications and extract a set of five application-level affordances (e.g. defining traits) that characterize WILD learning environments.

In the years since the “WILD article” was published, many scholars have contributed to the conversation around WILD learning environments by designing new WILD applications, better shaping the boundaries of this learning technology (e.g. Penuel et al., 2004) and contributing research around WILD applications in several fields of Computer Supported Collaborative Learning (CSCL) such as knowledge building and classroom orchestration. Looking back, it becomes evident that Roschelle and Pea correctly anticipated the evolution and growth of WILD learning environments. Macroworlds too “stand on the shoulder” of WILD and some of the application-level affordances proposed by Roschelle and Pea still present significant technical challenges to their effective implementation in macroworld applications.

The goal of this chapter is to survey a number of early macroworlds applications, similarly to what Roschelle and Pea did for WILD, in order to extract a set of five application-level affordances that are beginning to emerge and characterize these learning environments. I will then describe the challenges that each application-level affordance imposes on technology and derive a set of capabilities that a framework supporting the construction and enactment of macroworlds should have.

3.1 Support multiple macroworld simulations types

As described earlier, macroworlds' main goal is to allow students to access, manipulate and experience a simulated scientific phenomenon in order to study it while engaging in authentic scientific practices. In this context, one of the application-level affordances critical for the success of macroworlds is the ability to simulate a variety of different scientific phenomena. In particular, it is possible to identify three separate categories of macroworlds based on the type of object of inquiry they provide access to. First, there are *simulated phenomena*. In this type of macroworlds, a simulation of a scientific phenomena is build by the application designers and accessed by the students in the classroom. Students accumulate data over time by studying the phenomena and they use these data to build and verify their theories and models of the simulated phenomena. Among the macroworlds surveyed in chapter 2, the Hunting of the Snark, RoomQuake, HelioRoom, WallCology, AquaRoom and EvoRoom all belong to this first

category. Second, there are *generated phenomena*. In this type of macroworld application, which includes participatory simulations, BeeSim, the Hunger Games, the object of inquiry is represented by the data corpus generated by instrumenting students activities. In this kind of macroworld students act both as “agents” in the simulation that generates the data corpus and as “scientists” trying to understand the phenomena as a whole by studying and interpreting the data corpus. The third kind of macroworlds are the ones that allow students to engage with *emulated phenomena*. Designers of this kind of applications typically have a data corpus already available to them (e.g. historical records of some natural phenomena) or they have access to a real-time data source (e.g. instrumenting a scientific phenomena using sensors and telemetry). These macroworlds provide access to such data either by “replaying” the historic data in real-time or simply by streaming the data from the remote location to the classroom. This allows students to access the very same data scientists wrestled or are wrestling with.

Despite the value of providing different types of simulations and, within the same type, a variety of different science domains, imposes radically different demands on technologies powering macroworld applications. In simulated phenomena, for instance, designers need to be able to actually define somewhere in the software the rules that govern the simulation. These rules are unique, of course, for each simulation. Moreover, from an hardware perspective, some simulations are extremely demanding, requiring systems to track students movement (like generated phenomena) while others

simply require only a handful of computers (like HelioRoom for instance). This variety of requirements needs to be addressed somewhere, either on a “per-application basis”, leaving the burden on the designers, or harnessed in a software framework which can then be reused, freeing designers from solving the same problems over and over.

Capability 1: provide support for the creation and enactment of simulated, emulated and generated macroworld simulations.

3.2 Support the leveraging of the physical space of the classroom

Another key characteristic of macroworlds is the fact that they leverage the physical space of the whole classroom for pedagogical reasons. As described in chapter 1, this trait provides a series of benefits for learners such as allowing them to engage in a spatially situated exploration of the object of inquiry, increasing the amount of social exchanges among them (and therefore multiplying the possibilities for learning), affording them more agency, and ultimately making a better use of the classroom space.

The importance of leveraging the physical space of the classroom for pedagogical reasons has also been highlighted by Roschelle and Pea. They found that almost all WILD applications augment and leverage the physical space “between the devices” as opposed to more traditional CSCL applications which leverage the space “inside the devices”. Moreover, the authors point out that this application-level affordance of WILD

learning environments (including macroworlds) focuses on embodiment and, in particular, on using the physical space to help learners "reconnect abstractions with embodied, physical and spatial explorations" (Roschelle and Pea, 2002).

In macroworlds, leveraging the physical space of the classroom comes in two slightly different flavors. In generated phenomena (e.g. participatory simulations, BeeSim and Hunger Games), information is overlaid on top of the physical movements of students, captured and stored by the system. The aggregate patterns of students' movement then become the focus of students' inquiry as they try to make sense of their collective behavior. In a way, students' movements are tracked and are used as an input to the macroworld simulation and these patterns represent the data corpus driving students' inquiry.

Conversely, in simulated and emulated macroworlds (e.g. RoomQuake, HelioRoom, WallCology and AquaRoom, the hunting of the Snark and EvoRoom) the physical space of the classroom is used to "index" the simulation. In other words, there is a mapping between the physical space of the classroom and the virtual space of the simulated phenomena. In particular, the location of devices within the physical space of the classroom is relevant because it provides access to different portions of the simulated scientific phenomena. In this second scenario, students' ability to freely roam the classroom is crucial because it allows them to autonomously select and study different portions of the simulation (i.e. the different devices at different locations)

providing them with different points of view which they have to collectively reconcile, like in the story of the blind men and the elephant.

Despite the fact that leveraging the physical space of the classroom is one of the characteristic elements of macroworlds, this application-level affordance presents a series of practical challenges that need to be addressed in order to effectively employ it in the design of macroworlds. As mentioned earlier, macroworld application leverage a vast array of technologies and devices such as large (multi-touch) displays, laptops, tablets, smartphones, tangibles, iBeacons, etc. These differences among devices impact their capabilities in terms of their ability to sense their own location, with some devices reacting to proximity (e.g. iBeacons), others capable of more sophisticated location tracking and others incapable of both (e.g. large screens). Some devices are even capable of sensing their orientation, are frequently manipulated and moved (such as handhelds and tangibles) while other never change their location during an enactment of a macroworld-based curriculum unit (e.g. workstations). Not all macroworld applications impose the same requirements of spatial fidelity with some application requiring a very fine-grained positioning of devices within the room (e.g. RoomQuake) and others imposing very little requirements on the precision of locational information. In addition to these differences, some applications are based on the relative position among devices (such as participatory simulations) while other on the absolute position of devices with respect to the classroom space (such as embedded phenomena). Finally, different

devices have different network capabilities which render the distribution of locational information between devices and other components in the macroworld application even more challenging.

As it is easy to imagine, leveraging such variety of devices and variables (i.e. device type, sensing capabilities, granularity, degree of dynamism, relative vs absolute...) makes the macroworld application-space rich but, at the same time, hard for developers to leverage the physical space of the classroom. In order to successfully create a macroworld application that does this a developer needs to understand this vast application-space which in turn requires a significant effort and expertise. Therefore, any software tools supporting the construction and enactment of macroworld applications should provide a method and a set of tools to harness this complexity and make it more manageable.

Capability 2: support activities requiring location tracking and location awareness of significant numbers of objects and/or individuals across spectra of granularities and technologies.

3.3 Provide feedback during and after the enactment of macroworlds

So far I only talked about the “simulation” portion of macroworlds. However, the purpose of macroworlds applications is to enable the creation of a data corpus to be

used in subsequent knowledge construction activities. In macroworlds, such data corpus is created either by providing students with a set of tools to collect data about the macroworld (like in simulated and emulated phenomena) or automatically capturing and synthesizing their movement patterns (like in the generated phenomena) when acting as agents in an agent-based simulation (such as the Hunger Games or BeeSim). Independently from the fact that students are contributing to the data corpus by entering data with their fingers on a data-collection interface or using their body, macroworlds need to support this data collection process. One of the key characteristic of macroworlds is their ability to support students and teachers during this process by providing them (and researchers as well) with real-time and a-posteriori feedback. The next three sections will describe three separate scenarios for this application-level affordance: providing students with real-time feedback while building a data corpus, providing teachers with real-time preferential feedback over the data-collection process, and providing researchers with preferential, a-posteriori feedback.

3.3.1 Providing feedback for students and teachers

Roschelle and Pea (2002) argue that a key application-level affordance provided by WILD learning environments (including macroworlds) is their ability to aggregate the work of all students (not only a few) in real-time (not a-posteriori). They also argue that this aggregation process often results in a “coherent representation that can be read

and understood as a whole fairly easily” that can serve to direct the inquiry process as the students “see what they are building together”. The importance of real-time, aggregate representations of the data corpus as a way of directing the collective inquiry process, highlight areas of disagreement and serve as a basis for consensus has been reiterated also by Cober et al. (2012).

In science spaces, geo-spatial and semio-spatial representations (Roschelle and Pea, 2002) are often used to create shared, aggregate representations of the data corpus created by students. Geo-spatial representations are spatial graphical representations of spatial data (e.g. locations of students among each other or within the classroom) while semio-spatial representations are graphical-spatial representation of non-spatial data (e.g. a scatter-plot or any other graph).

An example of real-time, geo-spatial, aggregate representation of the students’ collected-data (i.e. the data corpus) can be observed in RoomQuake. As part of the unit, students were required to determine the position of a series of quakes. After each quake, students could use a data-entry interface available on their iPads (figure 1) to enter their reading of one (or more) of the four seismograms available to them as part of the simulation. As soon as students entered their observations, these observations were aggregated on a shared display available to them at the front of the classroom (figure 2). Using this shared representation, students could not only make sense of the data they

collected so far (helping them determine magnitude, time and epicenter of the quake) but also help them guide their future inquiry steps (i.e. which seismogram to read next).

An example of semio-spatial representation, instead, can be observed in HelioRoom. In HelioRoom, students's main goal is to associate planets names to the

The screenshot shows a web browser window with the URL `localhost:57880/roomquake/default/runs/rq-studen`. The page title is "RoomQuake Observations Entry". The main header features the "RoomQuake" logo with a red seismograph line and a target icon. Below the header, the text "Section: 6LM Student: BOZ" is displayed.

The interface is divided into two main sections: "Seismograph:" and "Calculations:". The "Seismograph:" section includes a dropdown menu with the value "1". The "Readings:" section contains three input fields: "P wave arrival time" (Sat, 30 May 2015, 20:09:25.5), "S wave arrival time" (Sat, 30 May 2015, 20:20:29.0), and "Maximum S amplitude" (125). The "Calculations:" section includes four input fields: "Difference in arrival times (S-P)" (3.5), "Distance" (5.25), "Magnitude" (4), and "Time of event" (Sat, 30 May 2015, 20:20:22.0). At the bottom, there are "Submit" and "Reset" buttons.

Figure 1. Tablet interface used by the students to collect data about simulated quakes in RoomQuake. Students use this interface to record their observations while reading the simulated seismographs in their classrooms.

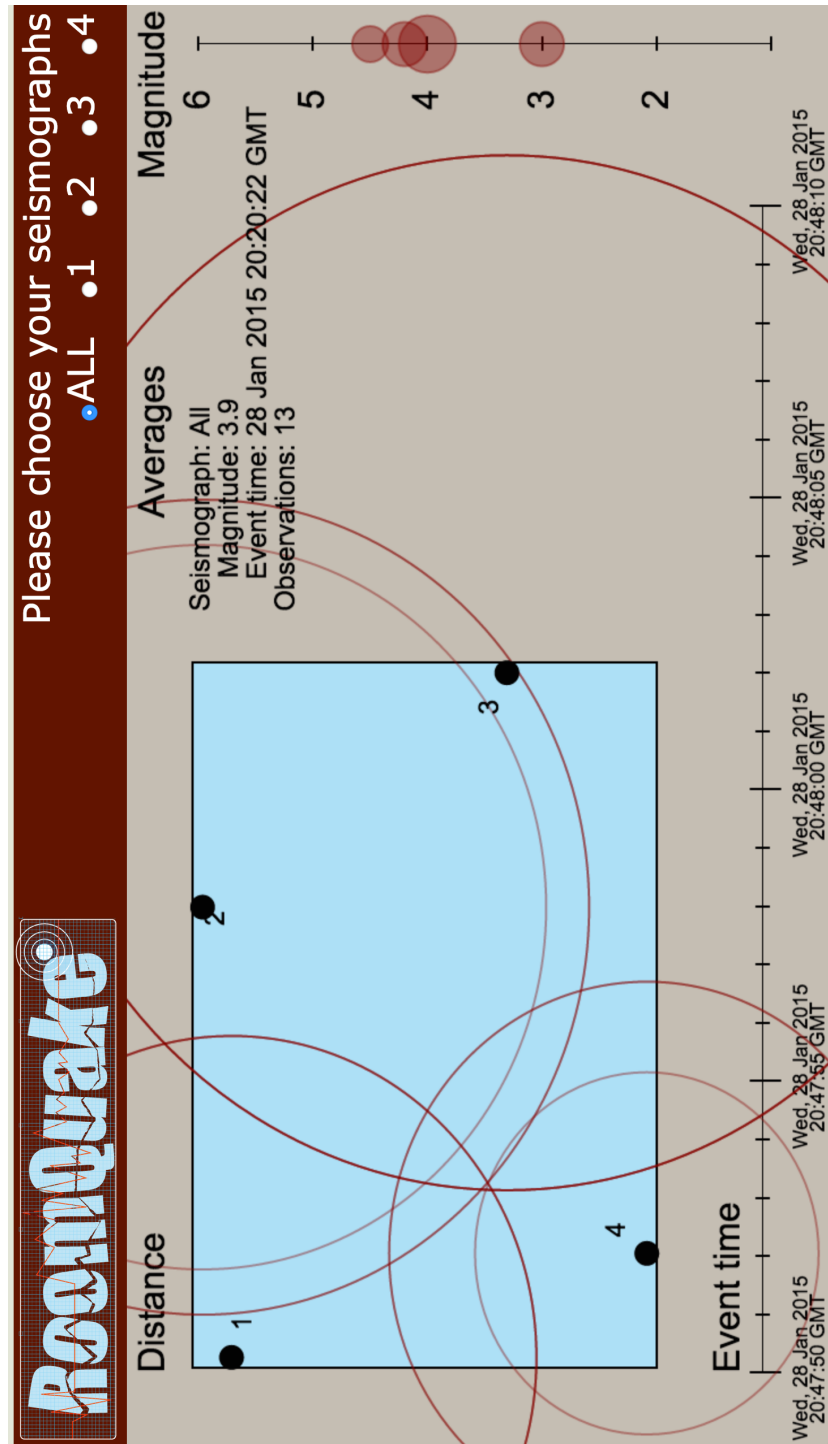


Figure 2. Aggregate representation of students' observations in RoomQuake. As students collect observations (using the interface in figure 1) about different RoomQuakes, the aggregate representation updates in real-time. As more students contribute data, the aggregate representation facilitates the students' discussion around location, magnitude and time of the event.

colored circles spinning around their classrooms and representing the planets in the solar system. In order to gather evidence to support this task, students collect pair-wise observations about planets' overlap (e.g. "red is in front of green"). In order to facilitate this process, designers of the HelioRoom application created an interface to allow students to collect these observations and, at the same time, provide them with feedback on the state of the inquiry process and the data corpus (figure 3). In addition to tallying students observations, (i.e. how many people observed that a planet is closer to the Sun than another) this interface also provided students with a list of people that agreed with a certain observation they made.

3.3.2 Providing preferential feedback to teachers

In addition to the feedback provided by the aggregate representations described above, sometimes teachers need preferential feedback on the data collection process in order to monitor student progress and provide them with additional formative feedback. For instance, simulated macroworlds are often designed to present an extended narrative, gradually revealed through the accumulation of students' observations over time. Therefore, the clarity of this narrative is highly sensitive to students' consistent and accurate data collection procedures. When erroneous data are added to the historical shared representation of the phenomenon, they can serve to obscure the underlying data-driven narrative of the unit. Therefore, the timely detection

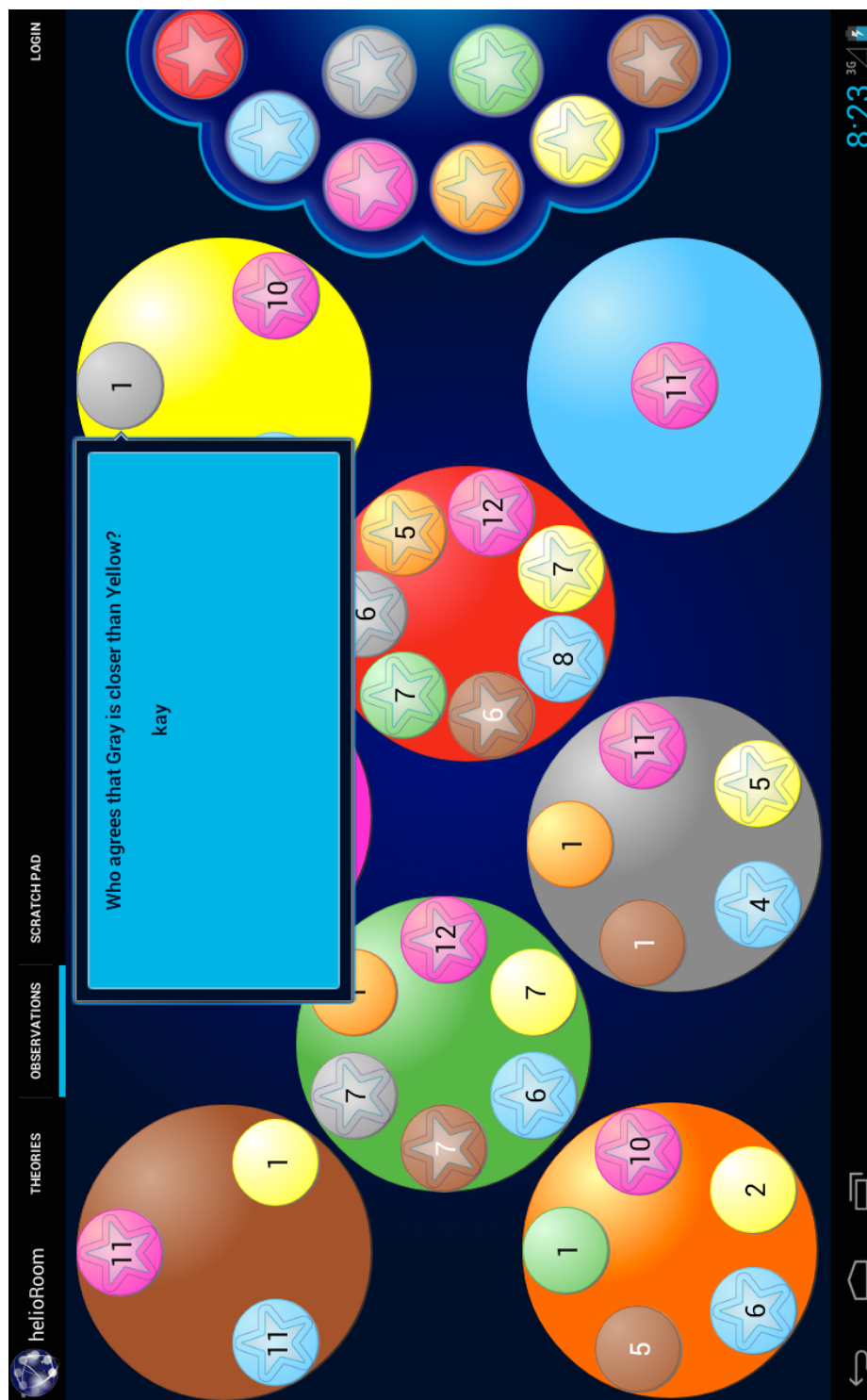


Figure 3. Aggregate representation of students' observations in HelioRoom. This interactive representation serves a dual purpose: it aggregates all students' observations about the planets they are trying to identify and it allows students to enter their observations. This is done by dragging small planets in the palette on the right, over to the big circles in the main area of the interface.

and remediation of practice errors become critical capabilities in successful enactments of macroworlds curriculum units.

An example of this can be observed in WallCology where teachers were provided with a tablet application that informed them in real-time whenever students' population estimates deviated from the values programmed into the simulation by more than 5% (figure 4). Using this strategy, we were able to reduce the error rates in students' population estimates from 33% to 8% (Gnoli, 2012).



Figure 4. Detail of the teacher feedback interface in WallCology. The interface notifies teachers of discrepancies between the creature counts recorded by students and the ones provided by the WallCology simulation. Teachers can take action upon receiving such notifications or dismiss them if they can't deal with them at the moment.

3.3.3 Providing a-posteriori feedback to researchers

Providing a-posteriori feedback on the data-collection process to researchers is crucial in order to enable them to use such data as evidence in their research. For this reason, macroworlds application logs are a common source of evidence used in research on this topic.

The importance of “log files” and application logs for research on macroworlds was, once again, outlined by Roschelle and Pea (2002). The scholars highlight how the ability to instrument the learning space with the goal of collecting summaries of messaging patterns and content for research purposes is another application-level affordance of WILD. Their intuition that “the database of interactions can be data-mined, analyzed, and reflected upon” has proven true as demonstrated by the nascent field of learning analytics and educational data mining. Accordingly, almost all macroworlds applications described earlier have been designed to collect and store messaging patterns between the various components of the applications and between all the devices used in the classroom.

3.3.4 Putting it all together

Providing students and teachers with real-time feedback on the status of the data corpus, teachers with real-time, preferential feedback on the data-collection process and researchers with a-posteriori, preferential feedback on the enactment of a macroworld unit imposes, once again, very different demands on technology. In particular, even if all three these scenarios are concerned with capturing messages between application components, there is quite a variety in the delivery strategies (e.g. push vs pull), the way these messaging patterns are accessed (e.g. real-time vs a-posteriori) and amount and type of filtering that needs to be performed. A framework supporting the

construction and enactment of macroworlds applications should support and harness this complexity and provide a mechanism to simplify the way developers handle these different messaging patterns.

Capability 3: support student's data collection process, including the ability of providing teachers, students and researchers with real-time and a-posteriori on their activity and the outcomes of their work.

3.4 Support classroom orchestration during the enactment of macroworlds

In both WILD applications and macroworlds, teachers can be metaphorically seen as "conductors of performances" to which each student in the classroom contributes to. Roschelle and Pea (2002) also suggest that teachers attend mostly to group performance (not individuals) and that they are responsible for choosing and sequencing the material introduced to students. Teachers might also send small groups to practice separately, as it happens in orchestra rehearsals, while making sure that all the orchestra sections are heard during a whole-class performance.

Virtually all macroworld applications surveyed earlier are designed thinking about teachers this way. For instance, in the WallCology enactment described by Cober et al. (2012), teachers were instrumental in guiding the process of inquiry, scaffolding whole-class discussions around aggregate artifacts, orchestrate (Fischer & Dillenbourg,

2006) and script (Kollar, Fischer & Slotta, 2007) the interleaving pedagogical, activities and simulation narratives. Findings about the pivotal role of the teacher as a “conductor of performance” in macroworlds (similar to the ones reported here for this particular enactment of WallCology) have been observed over and over in over a dozen macroworlds enactments.

The “conductor of performances” metaphor imposes significant enactment demands on teachers. They have to develop strategies to establish the scientific narrative driving the unit, and then strive to maintain it over the course of a multi-week, episodic time frame. They have to learn how to “use” macroworlds applications well enough to help the kids learn how to use them. Teachers have to engage learners in discussions involving question making, design and execution of investigations, data interpretation, and domain-specific practices. During the enactment of inquiry activities, teachers also have to monitor heterogeneous, physically distributed, often self-selected student activity and respond with appropriate pedagogical interventions.

The next four sections outline four common challenges faced by teachers (as “conductors of performances”) that I observed over the course of several enactments of macroworld-based curriculum units.

3.4.1 Provide teachers with privileged access, control and configuration over macroworld simulations

One of the key features of macroworlds is that the simulations are persistent and run continuously through the whole school day concurrently with the regular flow of instruction. In this context, key events might happen at times when kids can't divert their attention to the simulation and record their observations, missing on important learning opportunities. One of the major challenges faced by a significant number of the teachers who have enacted macroworlds units has been their inability to "tweak" the simulation (within the limits imposed by the laws of science) to ensure that all kids had a chance to contribute. The fidelity of the simulation to reality (i.e. nature doesn't wait for students) that was supposed to support deeper learning ended up undermining it. Teachers have been very vocal in the past about having affordances to make pedagogically motivated changes to the phenomena in response to students' behavior and needs. For instance, during one of the enactments of WallCology, teachers explicitly asked if it was possible to change the amount of creatures in the simulated ecosystem to make it more explicit for the kids that the introduction of an invasive species was decimating the population of a native one. In that particular situation, the teacher perceived that their kids did not see the gap in the native's species population as a first step toward their extinction.

The need for affordances that allowed teachers to gain privileged access to the macroworld simulation is a cross-cutting theme among many macroworld applications and in order to support teachers in their “conductor of performance” role several macroworlds applications provide technology specifically designed for this task. In one enactment of HelioRoom for instance, I developed a teacher dashboard application which was affectively called “the HelioRoom god view” (figure 5). This interface provided teachers with control over the simulation and with privileged access to the state of the simulation, beyond what was revealed to students through the portals. This allowed the teachers to validate students’ observation and better scaffold their reasoning. This Android application disclosed to teachers the position of all the planets in the simulation, a cheat-sheet revealing the association of planets and colors, together with their orbital time. Using this interface, teachers could also manipulate the position of an arbitrary planet and “pause” the simulation.

In EvoRoom (Lui & Slotta, 2014) instead, teachers have available to them a special tablet which allows them to “accelerate” the simulation in order to keep the phenomenal narrative aligned to the pedagogical and instructional narratives. Similar “macroworld simulation control dashboards” have been provided to teachers in other macroworlds, including the Hunger Games and RoomQuake (figure 6).

In particular, this interface allows teachers to configure and schedule a series of quakes in advance in their classroom. This is different from the real-time controls

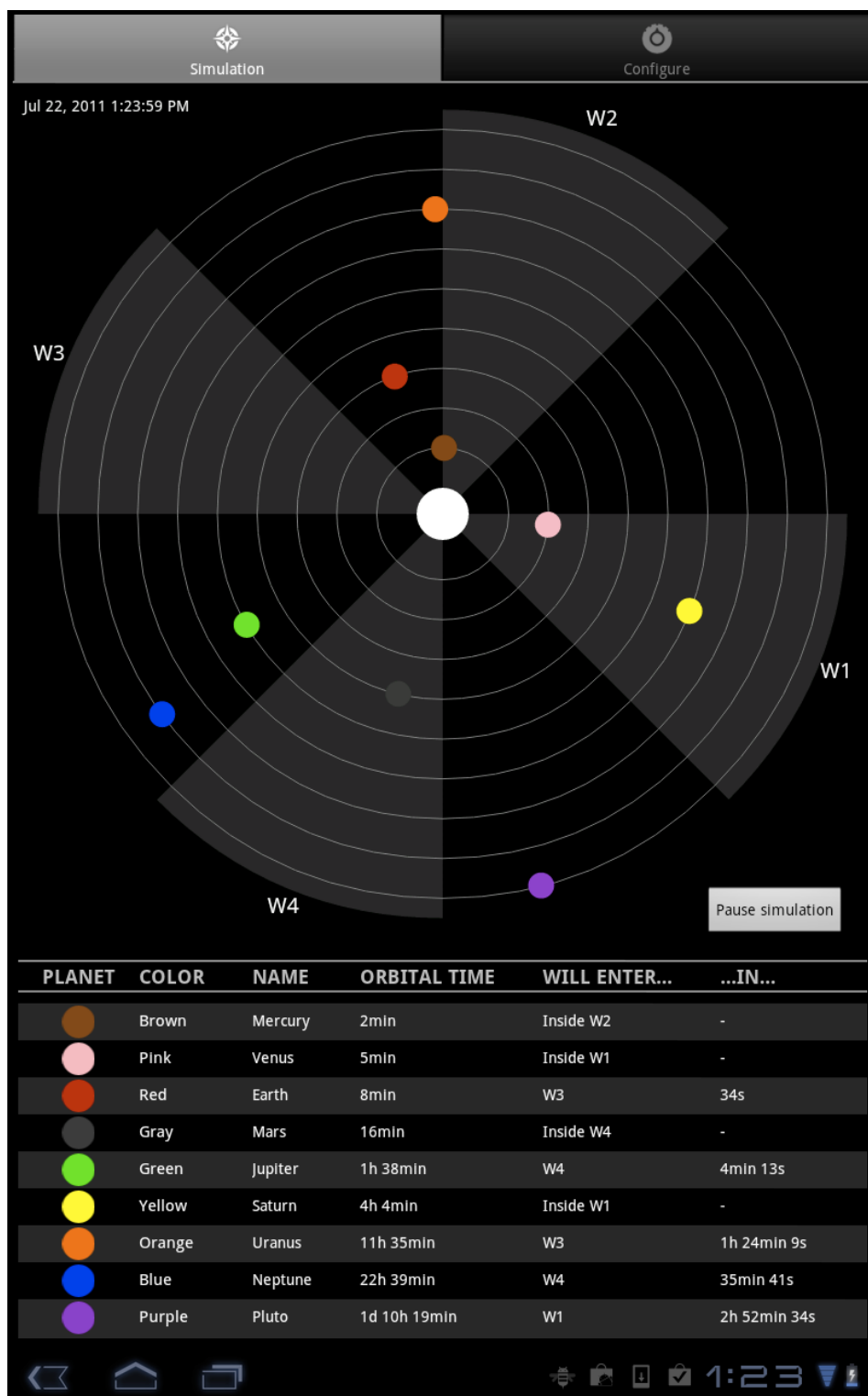


Figure 5. HelioRoom simulation control dashboard. This interface provides teachers with preferential information on the HelioRoom simulation. In particular, it discloses the position of all the planets and provides teachers with a cheat-sheet that associated the colored circles displayed by the simulation with the names of the planets.

provided by the HelioRoom and EvoRoom but it highlights how, for some macroworld simulations, it might be necessary to “schedule” events offline and in advance. This demonstrates how, despite several attempts have been made in the past to build teacher dashboards for the purpose of controlling and accessing macroworld simulations, each application is slightly different and requires a custom-made dashboard. Nonetheless, a

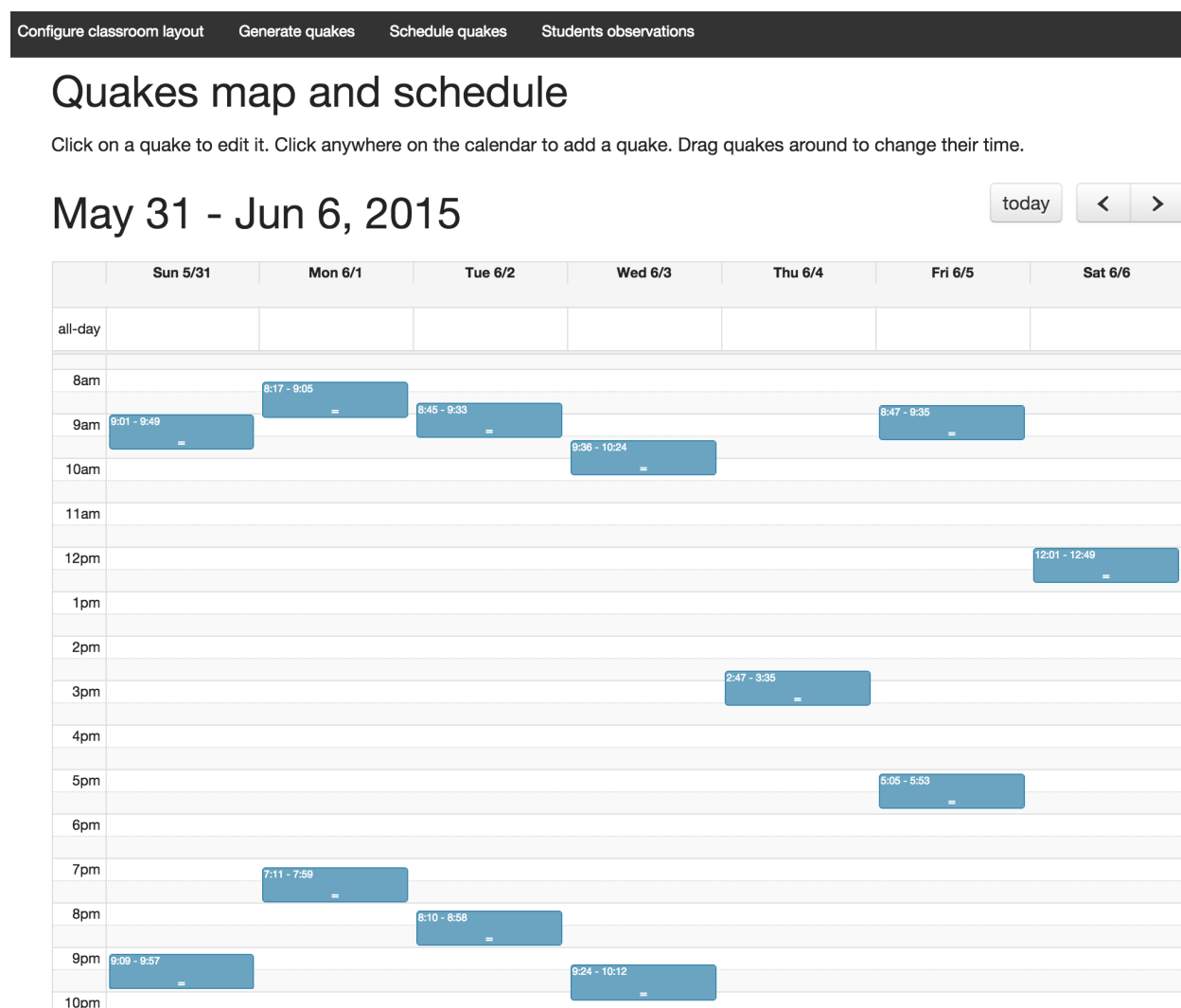


Figure 6. RoomQuake administration dashboard. This interface provides teachers with complete control over the simulation portion of the macroworld. It allows teachers to generate and manipulate a series of quakes that will unfold in their classroom.

framework supporting the construction and enactment of macroworld applications should strive to provide support for the creation of such utilities providing privileged access and control of the macroworld simulation to teachers.

3.4.2 Provide teachers with control over representational affordances, both public and private

As described earlier, macroworld simulations are used by students to create a data corpus. This requires students to collect data about the simulated phenomena. Sometimes this process is automated (e.g in the Hunger Games) and the results of this data collection are displayed to students. Either way, the representational and data collection affordances used by students need to be aligned with the simulation.

For instance in the most recent enactment of RoomQuake (Fall 2014), the curriculum was divided into two separate phases. During the first phase teachers generated a series of “on-demand” quakes and guided students through the process of understanding the procedures scientists used to study quakes and determine the position of the epicenter, the time and the magnitude of the quake. In the second portion of the curriculum, students studied a series of quakes and tried to determine the position of the fault line in their classroom and the distribution of quakes magnitudes, following the Gutenberg–Richter law. During these two phases of the activity the affordances used by students to collect their data changed dramatically and they

needed to be aligned with the simulation phases. This was achieved by making all affordances respond to a toggle button which switched between the modes (figure 7).

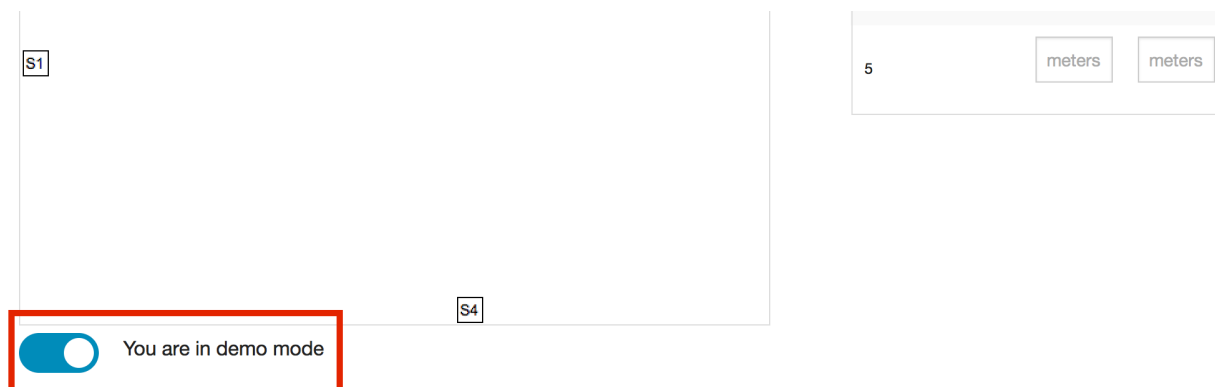


Figure 7. “Demo” and “schedule” mode switch for RoomQuake. Using this tool, teachers can choose to operate the macroworld in demo mode (where quakes can be generated in real time) or in schedule mode (where a series of quakes is scheduled for the future).

Another example of providing teachers with controls over representational affordances are the controls for the RoomQuake aggregate representation of students observations (figure 2). This interface was capable of displaying only one quake at the time so before students could move on to the next quake teachers had to “wipe the board” and prepare it for students to enter data about the new quake. For this reason, we provided teachers with a control interface (figure 8) that they used to change the quake students were working on.

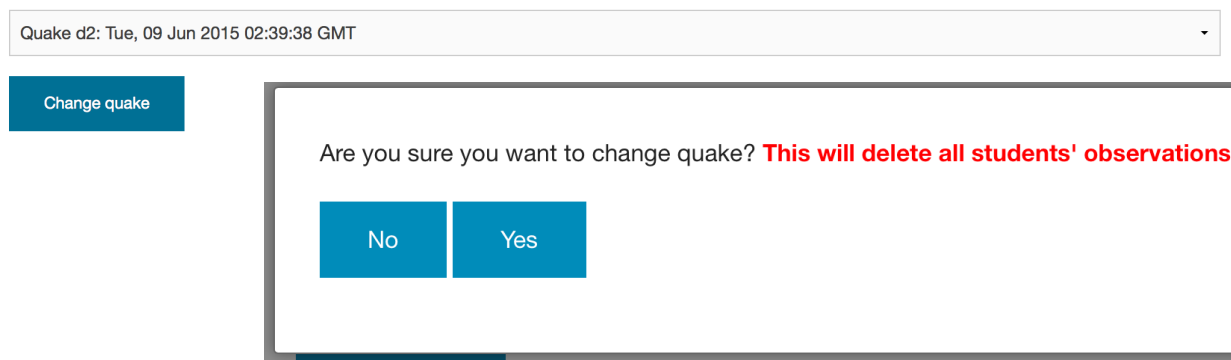
The point of both interfaces presented above is, representational affordances need to be controlled by teachers and often they need to be synchronized to the unfolding of the simulated phenomena. Controls for teachers to retain control over

representational affordances should be included in a framework supporting the enactment of macroworld applications.

Students observations

Students are currently working on quake **d4** that happened on Tue, 09 Jun 2015 11:50:23 GMT.

If you would like to change quake (and delete all students' observations relative to the current quake) select one from the list below and click on the "change quake" button.



Quake d2: Tue, 09 Jun 2015 02:39:38 GMT

Change quake

Are you sure you want to change quake? **This will delete all students' observations**

No Yes

Figure 8. Representational affordances controls in RoomQuake. Using this interface (top), teachers can control which quake students are working on. This automatically erases observations from previous quakes (popup), resets the aggregate representation and instruct the simulation bot to annotate students' observations with the quake number.

3.4.3 Provide a way to adapt macroworld applications to the technologies available in the classroom

When deploying macroworlds in real classrooms there is a huge variability among classrooms and schools in the technology available. This imposes on macroworlds an additional requirement of flexibility requiring a variety of progressively advanced classroom setups with some enactments deploying a very thin

layer of technology (Colella & Borovoy, 1997) and others taking full advantage of the technological affordances available. In addition, other factors such as teachers' confidence with technologies and the research goals of the enactment, have influenced the choice of technologies powering macroworld applications. Kids forgetting their personal device, collective devices breaking or being shared among classrooms are common place in most schools and macroworld need to be able to adapt, to a certain extent, to these daily disruptions.

3.4.4 Provide support for different instructional organizations

Another area where macroworld applications are required to be flexible is instructional organization. Some schools for instance have self contained classrooms, where students spend their entire day in the same room, while others have rotating classroom, where students change room at every period. Moreover, the same teachers might have multiple, concurrent sections engaged in the same macroworld unit. Other times, like in the last enactment of RoomQuake (Fall 2014), different sections might share the same macroworld simulation but collect separate datasets. Macroworld applications need to be able to adapt to all these scenarios.

3.4.5 Putting it all together

In summary, a framework supporting the enactment of macroworlds, should strive to:

- provide teachers with tools that afford them privileged access, control and configuration of the macroworld simulation;
- provide teachers with tools that afford them control of the representational affordances, both public and private;
- provide a way to adapt macroworld applications to the technologies available in the classroom;
- provide support for different instructional organizations.

Capability 4: provide teachers with support for classroom orchestration during the enactment of macroworlds, including controls for the macroworld simulation, adapting to instructional organizations and flexibility utilizing available technologies..

3.5 Support interoperability with other learning technologies

As already discussed, the goal of Macroworlds is to provide a way for students to engage in authentic scientific practices and generate a data corpus. However, despite the fact that data collection and participation in the scientific practices are important, in order for students to actually learn from the data they collected it is essential for them to

step back, reflect on such data and move away from rote “procedural engagement” (Gresalfi, et al., 2008). For example, while the individual mastery of population estimation methods in WallCology, is an important skill, it is intended to serve the larger objective of providing a base of data that students can use to collectively reason about the course of the phenomenon and build their understanding of ecological principles of population dynamics, habitat fit, and the impact of invasive species.

Reflection has been described as a primary means of social-constructivist learning (Bransford et al., 2000), as it allows students to build their own personally relevant understandings from their educational experiences. In the context of macroworlds, reflection becomes the collective work of a knowledge community (Scardamaila and Bereiter, 2003), in which the goal is to make progress on ideas based on contributions from all members of the community. During macroworld-based curriculum units, students are given responsibility over the whole knowledge construction process: generating new ideas and theories, building on other students’ ideas, and synthesizing ideas into higher-level concepts. The product of knowledge building activities is the creation or modification of public knowledge, which means that the collectively generated knowledge is available to other students and groups to be used, improved and worked upon. In this scenarios, students work collectively to develop a community knowledge base and improve upon one another’s ideas. This

process, in turn, helps students to develop a deep, personal understanding as autonomous learners (Bereiter & Scardamalia, 1989).

To support this knowledge construction process, several knowledge building systems and software infrastructure have been devised and developed. Of particular relevance here, is the SAIL Smart Space architecture (Tissenbaum & Slotta, 2015) an open-source software framework explicitly created to support Knowledge Community and Inquiry (Slotta and Najafi, 2010) which specifies a set of design principles for a knowledge community approach for science learning.

In order to make this knowledge-building approach possible in macroworlds learning environments, it is necessary for technologies powering the macroworlds simulations and the data collection infrastructure to integrate and interoperate with knowledge building systems such as SAIL Smart Space. In addition to providing a way for the two systems to communicate, this also requires the two software infrastructures to be aligned on a number of issues, such as data models, users, authentication and the notion of a classroom run (i.e. instance). For this reason, a framework supporting the construction and enactment of macroworlds should provide a way to integrate with knowledge building systems.

Capability 5: provide a set of strategies to integrate with software modules external to the framework supporting the construction and enactment of macroworlds.

3.6 Provide support for “non-functional” capabilities

In addition to the requirements described above, macroworlds impose a series of “engineering” requirements on the technology that powers them. This in turn forces developers of macroworld applications to deal with such constraints and address the challenges that they impose. For instance, the variety of devices, technologies and platforms typically used in macroworlds comes with very different “hardware footprints” and capabilities, requiring the use of different development strategies and languages. This complicates the communication among these components, especially when considering the unreliable network capabilities in most schools. This ultimately results in additional work and, consequently, additional resources (i.e. developers) that might not always be available (Slotta et al, 2009).

Part of macroworlds’ developers job is also to deal with this kind of issues, which need to be tackled in order to successfully develop a new macroworld application. However, all these “engineering challenges” are not unique to macroworlds but they are shared with other ubiquitous computing applications (Raychoudhury et al., 2013), technology-enhanced learning environments (such as Knowledge Building Systems; Slotta & Aleahmad, 2009)), distributed applications and, ultimately, a consistent portion of modern software. Nonetheless, support for tackling this kind of requirements should be present in any software tool attempting to assist developers when building

embodied macroworlds. Borrowing from the software and requirements engineering terminology, I define this class of challenges “non-functional challenges” since they are the analogous of “non-functional requirements” in requirements engineering.

In this scenario, supporting developers while tackling these non-functional challenges becomes a necessary requirement for any framework attempting to support the creation and enactment of macroworlds. Two requirements, in particular, stand out: a) the need of developers for diagnostic tools while constructing and enacting macroworlds and b) the need of developers for a mechanism to package, share and reuse their work.

non-functional capabilities: provide developers with a way to inspect and diagnose macroworlds during their construction and enactment together with a mechanism for the community to leverage each other work by sharing macroworlds’ components and applications.

Chapter 4

Literature review

The idea of designing a software framework to support the construction and enactment of a particular type of application is certainly not new to the domain of macroworlds. In particular, in ubiquitous computing research there is a rich tradition of creating frameworks and middleware to help developers deal with the inherent complexity of ubicomp applications and their implementation challenges. The first part of this chapter presents a survey of application frameworks in several ubiquitous-computing research areas that are particularly relevant for this work. The second part of the chapter, instead, focuses on previous work done around supporting the application-level affordances of macroworlds described in the previous chapter.

4.1 Application frameworks in ubiquitous computing

Perhaps, the most prolific of the research areas around frameworks and middleware for ubiquitous computing is the one concerned with *general-purpose application frameworks*. These frameworks combine context-awareness and ambient intelligence with networking abstraction, devices abstraction, and other techniques in order to provide a single, consistent middleware for all kinds of ubiquitous computing applications. The first such framework to be developed was Carnegie Mellon's Project Aura (Garlan et al., 2002) that provided a series of operating-system-level abstractions, such as nomadic file

access, “surrogate client” servers and resource monitoring, to “shield” ubicomp application developers from low-level implementation hurdles and allow them to focus on higher-level application logic. A second notable example of “general purpose” ubiquitous computing framework is Gaia (Román et al., 2002). This framework provides a components-based, distributed “operating system” for active spaces (“confined regions of physical space containing physical objects, heterogeneous networked devices, and users performing a range of activities”) which provides a set of tools and services (event manager, context and presence services, space repository and context file system) available to active spaces applications developers. Finally, systems such as OneWorld (Grimm, 2004) and HOMEROS (Han et al., 2004) focus on providing device/component abstraction and automatic adaptability whenever new devices join or leave the application spaces they are subscribed to.

Research around general-purpose ubiquitous computing frameworks continued to be a prominent area of interest in the ubicomp community through the years. Recently, work in this area has been focusing on the development of platforms to support Internet of Things (Ashton, 2009) applications. Examples of work in this area are Magic Broker (Erbad et al., 2008), Magic Broker 2 (Blackstock et al., 2010) and the latest iteration, ThingBroker (Perez de Almeida et al., 2013). This family of software platforms is designed to offer a simple and consistent programming interface to designers of IoT applications. The way these frameworks achieve this, and the main

contribution of this work, is a solid, event-based, publish/subscribe message passing middleware, which allows different components to communicate together. The authors also provide a set of APIs to abstract devices, sensors, and components and a facility to monitor all the communications mediated by the architecture.

Of course the frameworks described in this section represent just the tip of the iceberg of all the work done around applications framework by the ubiquitous computing research community. More thorough surveys exist and the reader interested in knowing more about this topic should reference the work of Raychoudhury et al. (2013), Bronsted et al. (2010) and Machado et al. (2013).

4.1.1 Context aware applications

One of the first examples of application-frameworks for ubicomp applications is Salber's et al. (1999) context toolkit that simplifies designing, implementing, and evolving *context-aware applications*. Context here covers information that is part of an application's operating environment and that can be sensed by the application itself. This work, in particular, emphasizes the strict separation of context sensing and storage from application-specific reactions to contextual information, and this decoupling facilitates the construction of context-aware applications. Yau et al. (2002), provide another example of framework to help designers of context sensitive ubicomp applications: their work provides a system to automatically decide which components

of the applications are allowed to communicate with each other based on certain elements of context (i.e. location).

4.1.2 Smart houses and ambient intelligence

A second area that saw considerable work in the early two-thousands was the one characterized by frameworks and middleware concerned with assisting designers of *smart houses* (Helal et al., 2005), *ambient intelligence* applications (Aarts et al., 2001), smart (Lee et al., 2003) and active spaces (Roman et al., 2002). Applications in these domains typically seek to help users carry out their domestic chores and everyday tasks (e.g. cleaning, cooking...) by “automating” some or all aspect of the home environment. Two examples of frameworks designed to help developers of ambient intelligence applications are the Gator Tech Smart House framework and IDE (Helal et al., 2005; Yang et al., 2006) and the EQUIP framework (Åkesson et al., 2002; Humble et al., 2003; Rodden et al., 2004). The first is a comprehensive framework for abstracting sensors and actuators and encapsulating higher-level context information while the second is essentially a data-space library (i.e. an object-oriented tuple space), which aims to support cross-platform and cross-language data sharing between networked clients. Analogously to what these frameworks do for the home environment, frameworks such as BEACH (Tandler, 2001), powering the i-LAND/RoomWare project (Streitz et al., 1999; 2001), provide tools to simplify the design of ubiquitous computing applications within

smart offices and workplaces. The BEACH framework consists essentially of a communication architecture together with a device abstraction layer that simplifies the design of applications where multiple user interacting with different devices (such as PDAs, Tablets and tiled wall-displays) need to communicate among each other to allow, for instance, the creation of a seamless, collaborative interaction space among the devices.

4.1.3 Ubicomp ecologies

Another area that has been getting quite a bit of attention lately are *ubiquitous computing ecologies* (Greenberg et al., 2011). A notable example of an application framework in this area is The Proxemics Toolkit (Marquardt et al., 2011). The main design goal of this platform is to simplify access to proxemics information for developers and its distribution among all the components of an application. For example, using this toolkit, students in a graduate Human-Computer Interaction course were able to build a proxemics application that allowed two portable computers to share a drawing canvas. The level of sharing increased as the laptops got progressively closer to each other with the user gaining progressive awareness of each others' work up to a point when they were sharing the same drawing canvas.

Ensemble (Brenton et al., 2013) is a software architecture designed to encourage exploratory development of distributed, multimodal, tangible, collaborative

applications by isolating changes to one component from other components. Particularly interesting is the design decision of making “supporting prototyping” the main design goal for the platform. Using this framework, the authors were able to create an application called Grendl, which allowed several performers in a university laptop and portable devices orchestra to “play” together and create music.

4.1.4 Pervasive games

Before wrapping up this section on frameworks for ubiquitous computing applications, I would like to briefly discuss the work around the creation of software framework to assist designers of *pervasive games*. According to Benford et al. (2005) “pervasive games extend the gaming experience out into the real world—be it on city streets, in the remote wilderness, or a living room.” Among the projects that pioneered the field of pervasive games, Equator (www.equator.ac.uk, 2000-2006) and its successor IPerG (www.pervasive-gaming.org, 2004-2008) introduced a number of pervasive games such as: Can you see me now? (Benford et al., 2006), Uncle Roy all Around you (Benford et al., 2004) and Epidemic Menace (Lindt et al., 2007). A side product of this research, and the technology powering all these pervasive games, is the EQUATOR Component Toolkit (ECT, equip.sourceforge.net; Greenhalgh, 2002; Greenhalgh et al., 2004). The toolkit is essentially a tuple-space, loosely coupled architecture that allows distributed applications to run over multiple hosts each running on or more software

components that can either be simple pieces of code, representations of physical devices and sensors or graphical user interfaces for the users. The framework also provides a set of tools for scripting, (re)configuring and monitoring the running application.

MoWeT (Barchetti et al., 2009) is an indoor/outdoor hybrid framework to design very simple location-based applications. In particular, the proposed software is tailored for a particular category of location-based applications called context-aware tourist guides (Cheverst et al., 2000): mobile phone applications that display multimedia content (e.g. web pages, images, sounds and movies) whenever a user gets close enough to a pre-specified Point of Interest (POI). Despite the relative simplicity of the applications it tries to support and the fact that it tailors a specific hardware and software platform, the MoWeT framework gets credit for trying to solve one of the longest running problems in Ubiquitous computing (Varshavsky & Patel, 2009): an accurate, low-cost, easy to deploy, and ubiquitous location-tracking system.

TaggingCreaditor (Sintoris et al., 2014), like MoWeT, has also been developed in the context of pervasive games for cultural heritage sites. This framework supports the design of what its authors call linking games. In these games, players roam freely in the game space (e.g. a city center, a museum, an heritage site) and, whenever they reach a Point of Interest, they use their location aware mobile devices to link real-world elements around the particular location they are in, with concepts, ideas, or factual knowledge in the virtual world. What TaggingCreaditor does is basically offering a

graphical interface to game designers that simplifies the process of specifying pairs of real-world elements (identified by GPS coordinates pairs, QR codes and NFC tags) and digital media. Such pairs are then stored into a database and then used by the framework to validate players' own matches, assigning them score and, sometimes, directing them to the next point of interest.

Although not properly a framework, Treasure (Guo, 2012) is the only toolkit I am aware of that specifically supports the creation of indoor pervasive games (by the players themselves). This framework allows its users to create simple treasure-hunt-style games by (1) hiding a series of smart objects (i.e. regular objects equipped with ultrasonic 3D sensors and Crossbow MOTE sensors) all over the game space; (2) using a graphical interface to specify the rules of the game, such as what conditions need to be satisfied to end the game (i.e. players found the treasure) and what kind of hints (i.e. media) needs to be displayed whenever players find clues to help them progress toward the end of the game.

The last framework in our survey is perhaps the most complex and feature complete of them all. fAARS (Gutierrez et al., 2012) and its predecessor fAR-Play (Gutierrez et al., 2011) provide a platform for designing a particular kind of pervasive games which the authors call Mobile Augmented Alternate Reality Games. These games use mobile phones, augmented reality and QR codes to allow players to impersonate both physical characters (using their bodies, their location in the real world and their

phones) and virtual ones (controlling a virtual avatar very much like in a traditional video game). The platform achieves this keeping the virtual and real world synchronized by using the mobile phones both as smart sensors (GPS location, QR codes, user input) and smart actuators (Augmented Reality). In addition to introducing their own frameworks, the authors present a survey of pervasive games platforms (Gutierrez, 2012).

4.1.5 Impact of ubicomp application frameworks on nutella

nutella's design leverages the work in the area of ubicomp application frameworks reviewed above in several ways. The idea of connecting many different devices using a message broker was inspired by the research of Blackstock et al. (2010) on Magic Broker. nutella combines this approach to devices-connectivity together with the "devices abstraction" approach typical of early frameworks for context-aware computing and smart-houses (e.g. Gator Tech House). Both approaches rely on abstraction to manage complexity in the same way as nutella does. Another area where nutella leverages this work is location sensing and tracking. In particular, nutella's central mechanism of decoupling location sensing and retrieval is heavily inspired by the work of Salber's et al. (1999) that also uses decoupling to simplify context handling. Finally, nutella's approach to physical-digital coupling is similar to the one used in

fAARS (Gutierrez et al., 2012) where real and virtual worlds are coupled through the use of ambient and mobile technology.

Despite all the work in this area, however, there is no single framework or technology among the ones reviewed above that provides all the key technology capabilities of macroworlds. For instance, while Ensemble (Brenton et al., 2013) contains many of the elements that are also in nutella, it lacks support for location sensing and tracking, a crucial component to enabling macroworld design and enactment.

4.2 Supporting the construction and enactment of macroworld applications

In this section, I will describe work that has already been done around the challenges that arise when constructing and enacting macroworld applications.

4.2.1 Support multiple macroworld simulations types

To the best of my knowledge, there are no other systems other than the one presented in this dissertation that enable the design and enactment of macroworld applications based on simulated, emulated and generated phenomena. However, nutella evolved from previous work by the author himself on the *phenomena server*, a technology that provided support for the development and enactment of a particular kind of simulated phenomena called embedded phenomena (Moher, 2006). The phenomena server was an application container that provided encapsulation and a

common interface for different types of embedded phenomena simulations. The phenomena server enabled developers to create applications following a standard structure and a shared set of APIs, which simplified and unified different embedded application designs. The phenomena server proved extremely reliable and stable by powering dozen embedded phenomena classroom enactments with hundreds of days of continuous operation. It has recently been decommissioned because it has been replaced by the system described in this dissertation. However, some of the ideas and technologies developed during the work on the phenomena server continue to live on in nutella.

4.2.2 Support leveraging of the physical space of the classroom

As highlighted earlier, leveraging the physical space of the classroom, imposes a series of challenges that developers of macroworld applications need to leverage. The first challenges that developers of this kind of applications need to overcome are the limitations of today's indoor location-tracking and sensing technologies. Cheap, reliable and easy to install tracking systems for all individuals and relevant devices ($n=30$) in a classroom covering the whole classroom space doesn't exist today, to the best of my knowledge. Historically, designers solved this problem by asking students to either self-report their location (Novellis & Moher, 2011) or specifying it a-priori (Moher et al., 2005) through the use of a traditional desktop-based interface. In the last ten years

however, there has been tremendous progress in the affordability of systems based on RFID systems (e.g. Gnoli et al., 2014) and iBeacons (e.g. Sørensen & Kjeldskov, 2013), capable of detecting proximity of up to 30 objects among a number of “hotspots” scattered around the classroom.

At the same time, as described in earlier sections in this chapter, the ubicomp community has been hard at work building frameworks and middleware to provide developers with an easier way to access location and proxemics information. Unfortunately, almost all these frameworks are affected by one (or more) of the following. a) They are often (still) very tightly coupled with a particular kind of location awareness technology such as RFID, GPS, some “flavor” of tracking, etc. b) They often mix context and location awareness together with other features (such as context reasoning, storage, modeling, etc.) making them impractical to use (Buschmann, 2010). c) They provide support only for devices that are capable of sensing their own location frustrating the articulation of a single, consistent and coherent paradigm for location awareness.

4.2.3 Provide feedback during and after the enactment of macroworlds

In addition to the work reviewed by Roschelle and Pea (2002) and described in the previous chapter, Jim Slotta and his group have done a lot of work on providing co-located students with real-time feedback on their actions. In particular, his research in

this area focuses on students collectively developing a shared discourse that allows for idea-sharing, critique and improvement (Brown & Campione, 1996) in order to advance a shared knowledge base (Bielaczyc & Collins, 1999). To this end, Slotta and his group have developed a software framework called SAIL Smart Space (Tissenbaum & Slotta, 2015) which allows for more rapid development of learning materials and environments enabling a broad research program on collaborative inquiry.

Complementary to research programs exploring feedback to students and teachers alike, others have explored the possibility of providing teachers with preferential feedback on their students' activities using tangible technologies. FireFlies (Bakker et al., 2013) is a tangible learning technology that enables teachers to continuously gather feedback on student's activities using the periphery of their attention. Interestingly, the system comes with a tangible teacher tool, which enables the teacher to control the shared-ambient display and regulate the rate of feedback they get.

4.2.4 Support classroom orchestration during the enactment of macroworlds

There is a long tradition of research in the learning sciences on the most effective ways of supporting complex collaborative learning designs in the classroom. One common approach that is used to support the orchestration of complex collaborative learning activities is that of the "collaboration script," which has been shown to effectively foster collaboration and improve learning outcomes (e.g., Weinberger, et al.

2005; Rummel & Spada, 2005). However, of particular interest for this dissertation, is the use of technology for classroom orchestration. Research in this area has been pioneered by Dillenbourg and Jermann (2010). In their paper, the authors summarize 14 design factors (e.g. flexibility, control, integration) to take into account when designing technologies for classroom orchestration.

Few researchers in Human-Computer Interaction have also worked on tangible technologies for classroom orchestrations. As an example, CowClock (Bakker et al., 2012) is a tool that fosters time awareness in primary school classrooms. Teachers can mark timeframes in the clock using tangible tokens and the clock will play soundscapes associated to the token with increasing frequency to signal the passing of time. A qualitative, two-weeks exploration of CowClock in the classroom revealed that the peripheral, auditory feedback provided by the tool was successful in fostering time-awareness.

4.2.5 Supporting interoperability and reuse in learning technologies

The problem of reuse of software components in educational technologies has been around for as long as there has been educational software. In 2009, a panel titled “Toward a Technology Community in the Learning Sciences” was presented at the 8th International Conference on Computer Supported Collaborative Learning (Slotta et al., 2009). In their presentation, panelists highlighted several challenges in the learning

sciences community that hamper the design of reusable software components and the interoperability between technologies designed by different research groups. Quintana and Soloway highlight issues with the grain size of software components, their customization and usability of software frameworks used to stitch components together into applications. Roschelle and Pea lament how other developers and learning scientists are not using ESCOT, one of the first attempts at componentizing educational technologies, to build learning technologies. The reason for this, they thought, was due to the immaturity of the underlying technology platform at the time (Java) and the divergence in handhelds operating systems. Finally, Aleahmad and Slotta describe their experience successfully designing SAIL, an open-source platform for learning technologies. Their strategy was to “separate out the plumbing” from the content so it could be used in different applications.

Besides talking about components, software, and frameworks, the authors also make an important point about the importance of developers’ communities as a venue for developers to exchange ideas, work on collaborations and explore technologies. They also briefly outline their plans to create a learning-technologies, developers’ community, called Educoder (Slotta & Aleahmad, 2009). The importance of a community as a way to help foster the reuse and development of open-source tools that solve recurring problems is not an idea new to the learning sciences. Communities like

Github (<https://github.com>) have proven to foster developers' collaboration and advance the community (Dabbish et al., 2012).

Chapter 5

nutella

nutella (<http://nutella-framework.github.io>) is an open source framework that supports the building and enactment of macroworlds. This chapter, describes the internal software architecture of nutella, its main functional sub-modules, and provides a high-level overview of the process of developing and enacting a macroworld application using the framework.

5.1 nutella's architecture

nutella's internal design is heavily inspired by the actor model of computation (Hewitt et al., 1973). According to this paradigm, the basic unit of computation is an actor. An actor is simply a piece of software that executes on some host, can receive inputs and produce outputs. Complex behavior is modeled by composing many actors together to obtain the desired functionality. In nutella, consistent with this model, applications are collections of actors which can be of only two types: *user interface* or *bot*. User interfaces are actors which can receive inputs from other actors and human users (typically through a Graphical User Interface, GUI), while bots are actors that can only receive inputs from other actors. Actors communicate among each other using messages. Actors register event listeners for incoming messages (and user inputs if they are user interfaces) and emit messages as outputs. This “inputs, computation, outputs”

has been strongly influenced by the ubiquitous (pun not intended) Mode-View-Controller pattern (Krasner & Pope, 1988).

nutella's internal software structure is divided into three layers: existing software, APIs, and components, with the framework core running across the previous three layers (figure 9).

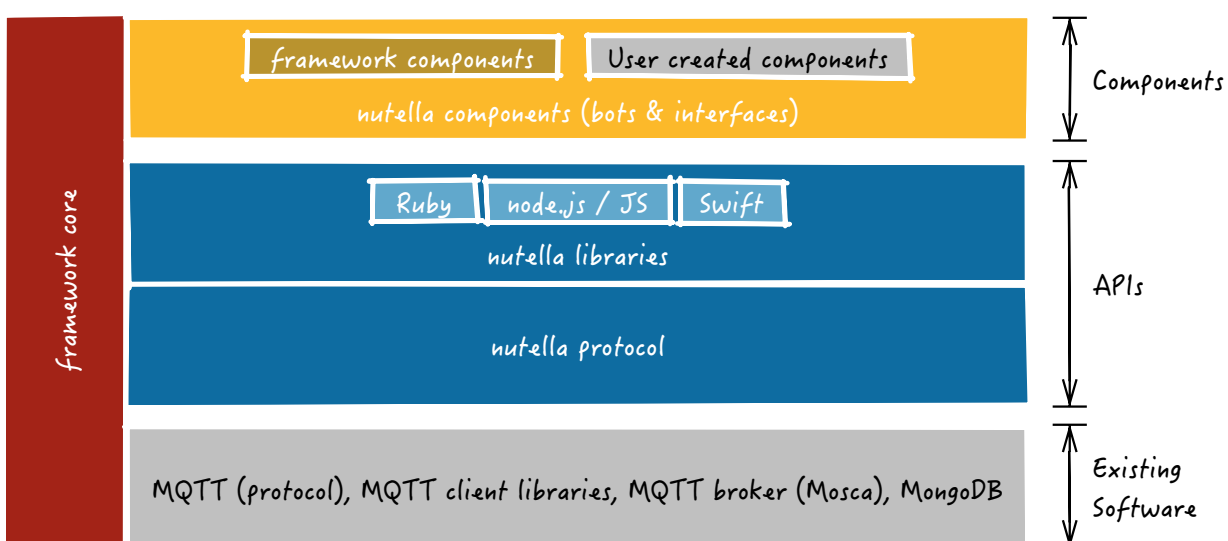


Figure 9. Overall architecture of the nutella framework. nutella's internal structure is organized into three vertical layers (existing technology, APIs, components) and a framework core perpendicular tying these three levels together.

5.1.1 Existing technology

At its core, nutella is powered by a robust message-oriented middleware (Curry, 2004) that allows actors (called *components* in nutella) to exchange messages among each other. In particular, all the messages exchanged by software components go through a

message broker, which, in the case of nutella, is an open source software called mosca (<http://www.mosca.io>). All exchanged messages must comply to the MQ Telemetry Transport (MQTT) protocol, which was designed by IBM as a lightweight publish/subscribe protocol specifically for Internet of Things applications (<http://mqtt.org>). This protocol was chosen because of its small memory footprint and simplicity that make it easy to implement it even on devices with very limited resources (e.g. Arduino, tangibles, etc.). Moreover, the established open-source community maintaining the protocol and its support for both WebSockets and regular TCP/IP sockets make it practically ubiquitous among platforms and programming languages, providing developers with a very high degree of freedom when choosing platforms.

5.1.2 nutella protocol

On top of this existing layer of technology sits the first abstraction layer provided by nutella, what I call the *APIs layer*. The goal of this portion of the architecture is to provide to the all the components in a nutella application a set of expressive primitives to communicate with each other. The goal of this layer is to move designers and developers away from low-level communication primitives such as “connect”, “re-connect”, “disconnect”, “keep-alive”, “set timeout”, etc. and provide them with a set of more expressive communication APIs.

The first step in this process is to define a communication protocol, that specifies the structure of the messages that are exchanged between components. I decided to use JSON (<http://json.org>) as a data interchange format because it is human readable (and therefore easily debuggable), open, widely adopted, and because it has a light footprint compared to other human-readable formats such as XML. Moreover, exactly like there are MQTT libraries for virtually any programming language, the same is true for JSON parser libraries.

The nutella protocol (<https://github.com/nutella-framework/docs/blob/master/protocol/index.md>) specifies the structure and semantic of the JSON messages that are sent back and forth between software components. In particular, the protocol specifies four primitives that actors can use to communicate with each other:

- Components can ask question directly to other components
- Components can answer questions other components ask them
- Components can say things to an audience of components that are listening and are interested in what the “speakers” are talking about
- Components can express their interest in what other components are saying, listen and wait for them to say something

These four actions can be grouped into two separate communication strategies: *request/response* or *pull* (first two actions) and *publish/subscribe* or *push* (last two actions). Each one of the two messaging patterns has its list of advantages and disadvantages

(Rodríguez-Domínguez et al., 2012) which I am not going to cover here. Nonetheless, I would like to provide some rationale for making both available in nutella.

The request/response (or pull) mechanism was born to allow communication in client-server applications. In this model, clients are responsible for visualizing data while servers are responsible for storing them. Clients request the data they need to display to servers that reply. In nutella applications, data is rarely centralized in a single component but it is instead distributed among different actors. This creates a situation where each actor is, at the same time, a client and server for different data. In this scenario, a request/response messaging strategy presents two main limitations. First, if an actor needs data that is stored within several other actors it needs to know what data is where and then go perform all the requests needed to retrieve such data. Second, it is impossible for an actor to just wait for data to be delivered to them, whenever it becomes available, but it forces the “client” actor to constantly *poll* other “server” actors, generating a lot of useless network traffic.

For this reason, the main communication mechanism provided by nutella is publish/subscribe. However, request/response is also provided because there are cases where a push mechanism alone can become cumbersome. For instance, consider the frequent scenario where an interface (client) is accessed and it needs to display data that is stored in a different bot (server). In this particular use case, Here, it is simpler to use request/response instead of: (1) having the interface listen for new data; (2) having the

bot listen for requests from the interface; (3) having the interface signal that they are up with a message; (4) having the bot capture that message (request), fetch the data and signal they have data (response); (5) having the interface capture that message (response) and use the data.

The limitations of pull strategies together with the corner-cases inefficiencies of push strategies are the main motivation behind the use of both messaging patterns in nutella.

5.1.3 Native language nutella libraries

The layer on top of the nutella protocol is a set of libraries implementing the protocol itself and simplifying the interaction with higher-level components of the framework. The rationale behind the choice of implementing native language libraries is rooted in the desire to reduce the number of entry barriers developers have to overcome in order to adopt the framework. While “implementing the nutella protocol in your favorite programming language” can be a fun exercise, it is an extra step that a developer needs to make in order to take advantage of the communication facilities and higher level function provided by nutella. Vice-versa, if the protocol is already implemented in a language-specific library, developers familiar with the language can import it and use it with the tools they are already familiar with.

In addition to implementing the basic nutella communication protocol described above, the nutella libraries also implement higher-level primitives to facilitate the communication with the framework components (see next section). Again, this simplifies developers' lives because they are not forced to implement the portions of the protocol that enable the communication with higher-level framework components.

So far I implemented native libraries for JavaScript (browser and node.js), Ruby, Swift. Java (standalone, Processing and Android) is in progress and I plan to extend the range of available languages in the future by implementing language-specific libraries for C# (Unity), Scala and Python.

5.1.4 nutella components

The last layer in the “nutella architecture sandwich” is the *components layer*. As described earlier, components (i.e. bots and user interfaces) encapsulate all the logic in nutella applications and communicate among each other using the native-language APIs provided by the nutella libraries. As highlighted by the diagram in figure 9, components come in two slightly different flavors: framework components and user-defined components. Framework components, as the name hints, are integrated with the framework itself and provide a set of high-level features (e.g. access to location, orchestration, logging...) to all nutella applications. These six framework components are the “heart” of the nutella framework and I will cover each one of them in detail in

section 5.2. User defined components, instead, are the way designers and developers implement macroworld applications. Each component implements a portion of the macroworld application and, together with framework components, they collectively implement the desired functionality.

5.1.5 Framework core

With the macroworlds' application logic distributed among actors, a crucial capability of the architecture, besides allowing communication between actors, is the ability to manage the components' lifecycle. Starting and stopping them, generating new actors and destroying them are tasks that are typically handled by "actors supervisors" (Hewitt et al., 1973). In nutella, this role is handled by a "vertical" module called *framework core* that spans across all layers of the architecture. The framework core provides a command line utility and APIs that can be used by developers to generate macroworld application skeletons, control and configure the runtime environment, instantiate new actors from templates and perform other "support" tasks during the development and enactment of macroworld applications.

5.2 framework components / macro-modules

As described earlier, framework components implement most of the high-level features provided by nutella. Since each framework component typically comes with its

own protocol extensions, and relative native-language library support, I am going to call the framework component, protocol, and native-language implementation set a *macro-module*. nutella comes with six macro-modules: RoomPlaces, RoomCast, RoomRecorder, RoomComponents, RoomDebugger, and RoomMonitor.

- *RoomPlaces* is a classroom, physical space, and resources (i.e. devices) manager.

It provides developers of macroworlds with a GUI that they can use to add/remove/edit devices used by their application and configure their location, including automatic location tracking. It also provides a set of APIs that can be used to access all information about resources' location and the classroom spatial organization in real time.

- *RoomCast* is a “cable TV system” for the classroom. Besides providing a way for students and teachers to access a macroworld application, RoomCast allows developers to create channels and channel packages (using a GUI) out of existing visualizations and user interfaces. RoomCast allows developers (also via a GUI) to configure which users are subscribed to which packages and store this information inside a configuration. Many configurations can be used at the same time and can be controlled using a mobile, “TV remote” app which can be operated by teachers.

- *RoomRecorder* is a distributed logging utility for macroworld applications. It automatically records all messages that are exchanged between all the components of a macroworld application and provides a mechanism to access and filter this

information both in real-time and a-posteriori. Under the hood, RoomRecorder is powered by a document-oriented database called MongoDB (www.mongodb.org).

- *RoomComponents* are a set of application components skeletons which can be used by macroworld developers to speed up their development. They are accessible in an online repository and any developer can contribute their component for others to use. This also provides macroworld developers with a way of reusing and sharing their work. At the moment of writing, there are 13 components in the repository.

- *RoomDebugger* is a debugging and testing GUI used by macroworld developers to mock application components (bots and interfaces) during the development process. This component is particularly useful to test the communication and interaction among components in an application.

- *RoomMonitor* provides developers with a GUI to monitor macroworld applications in real-time both during their development and enactment. Using RoomMonitor, developers can be notified via email when issues with macroworld applications arise. Moreover, RoomMonitor provides them with a way of progressively “drilling down” from all applications, to a specific application, to a specific instance of an application, to a specific component, helping them find the root cause of issues whenever they happen.

5.3 Macroworlds development process with nutella

The framework components described above, together with the rest of the nutella framework, were designed to support a particular development process derived mostly from my experience creating several macroworlds applications in the past seven years. In this section I will go through each one of the phases in the development and enactment processes of a macroworld application and show how each of the macro-modules in the nutella framework provides selective support for developers engaged in each one of these phases. In the next chapter, I will go into more detail and provide specific examples of how nutella addresses the challenges described in chapter 3. The overall development and enactment process enforced by nutella is depicted in figure 10.

5.3.1 Initial design

The first step in designing a macroworld application is to decide what to build. Typically, this involves a set of brainstorming sessions and several iterations where the design is progressively refined until it reaches a point where it is stable enough to start implementing it. It is important to point out that the design and development processes are tightly coupled and highly iterative. What is called initial design in figure 10 could be as simple as a paragraph-long description of the macroworld together with a set of ideas for its development.

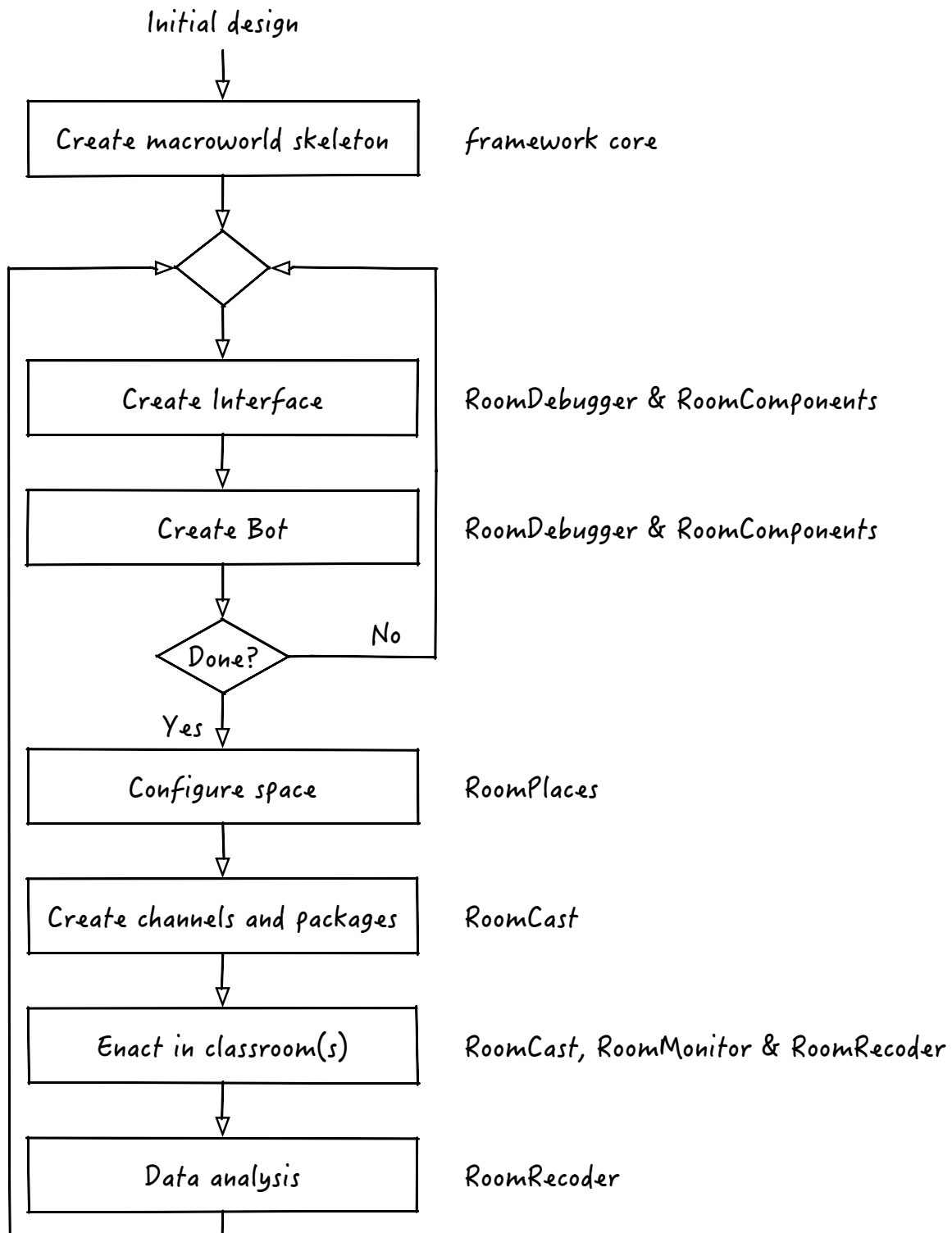
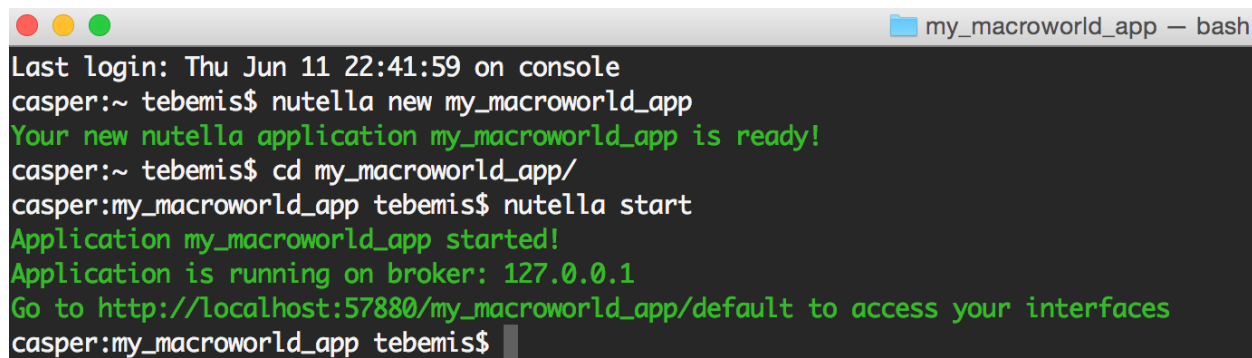


Figure 10. Steps involved in the construction and enactment of a macroworld application using Nutella. On the right, the name of the Nutella macro-modules supporting each step.

5.3.2 Macroworld skeleton

During this phase, developers use the Command Line Interface provided by the framework core module to create a skeleton of a macroworld application. However, developers need to install the framework on their computer first. In order to do so, they need to follow the instructions on the nutella website (<http://nutella-framework.github.io>) to a) install all the software prerequisites (node.js, Ruby, Git, tmux and MongoDB); b) download and install the framework using RubyGems (`gem install nutella_framework`) and c) initialize it (`nutella checkup`). Once developers successfully complete this procedure they will be able to type the command `nutella` in their shell and get a welcome message.

If `nutella` is already installed on their computers, developers only need to type three shell commands (figure 11) to create a new macroworld application skeleton that can then be customized and is immediately accessible via any web browser (figure 12).

A terminal window titled "my_macroworld_app — bash" with a dark background. The text is as follows:

```
Last login: Thu Jun 11 22:41:59 on console
casper:~ tebemis$ nutella new my_macroworld_app
Your new nutella application my_macroworld_app is ready!
casper:~ tebemis$ cd my_macroworld_app/
casper:my_macroworld_app tebemis$ nutella start
Application my_macroworld_app started!
Application is running on broker: 127.0.0.1
Go to http://localhost:57880/my_macroworld_app/default to access your interfaces
casper:my_macroworld_app tebemis$
```

Figure 11. Output of "nutella new" and "nutella start" commands. Using the nutella framework core developers can create and start a nutella application skeleton with few simple shell commands.

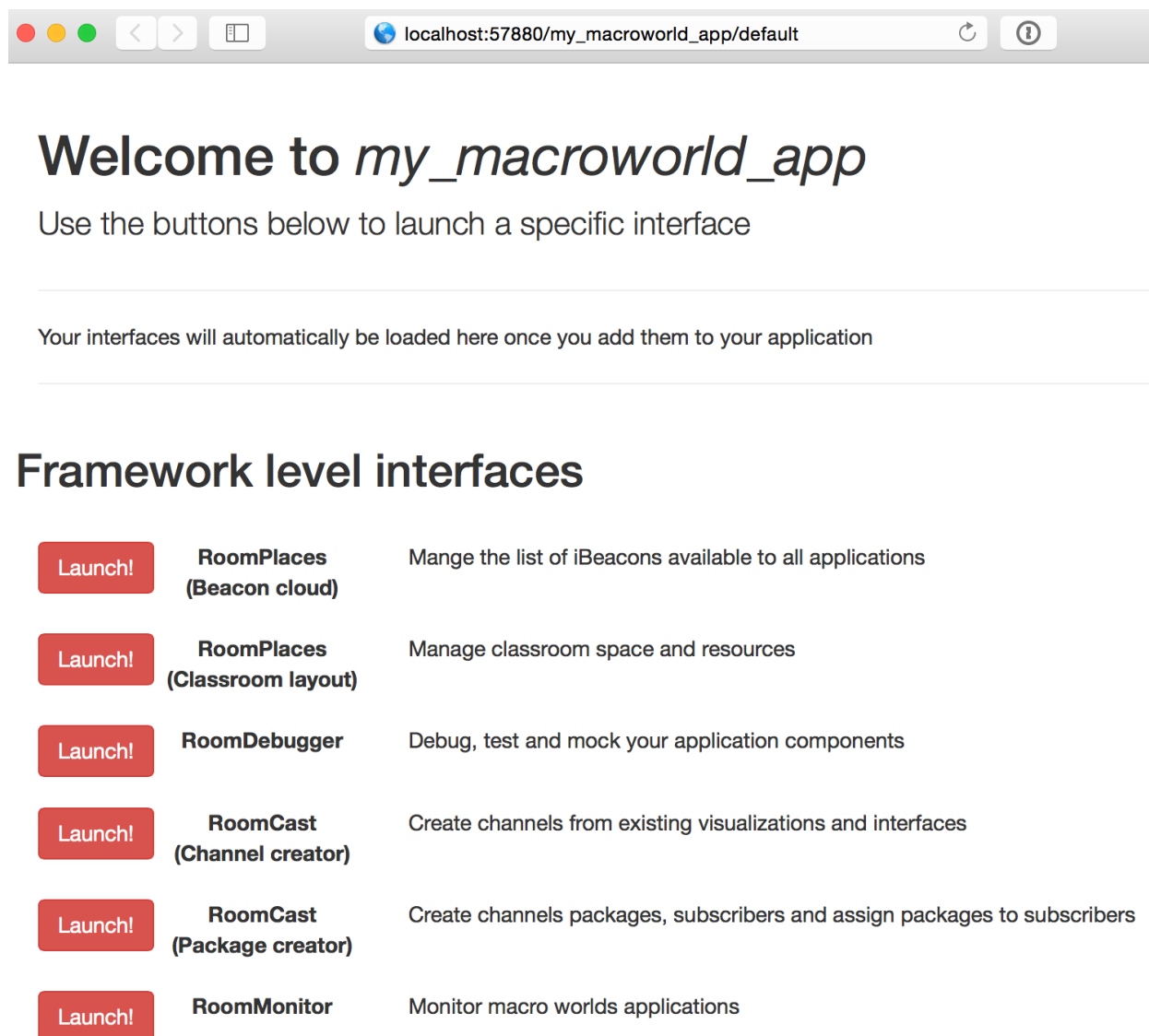


Figure 12. nutella's main interface. This is the default interface for a new macroworld application presented to developers. Developers can access almost all higher-level functions provided by the framework using the buttons in the lower portion of the interface. As new interfaces are added to the application they become automatically accessible via this interface.

5.3.3 Create user interface(s) and bots(s)

With an application skeleton ready to be customized, developers can move onto the creation of user interfaces and bots. In my experience, most developers tend to start

with user interfaces but there is nothing in the framework preventing them to start from bots. nutella treats both user interfaces and bots as components.

The first step a developer should take when creating a new component is to use the repository of RoomComponents to see if someone else has already created a component template that would simplify the component creation process. Most RoomComponents provide some type of boilerplate code (in addition to the language specific nutella library) which provides a way to kick-start the development, speeding up the process of creating a new component. Examples of RoomComponents are, for instance, web user interfaces built using a particular framework (such as Facebook's React.js), an iOS visualization, a Processing application, a simulated phenomenon, etc. One of the main advantages of RoomComponents is that they don't place any restriction on the granularity of templates with some providing minimal support while others can be fairly complex and complete. This provides developers with a flexible array of components' templates to choose from, reducing the amount of boilerplate code they need to write and allowing them to focus on functionality instead. Once developers have identified the right template they can install it in their nutella application/project with a single command and start modifying it to fit their needs.

As describe earlier, components don't exist in a vacuum but they all work together to create a macroworld application. However, it can be challenging to test communication among components when only some of them have been created. For

this reason, nutella provides a tool (called RoomDebugger) to simulate and mock components. Using this tool, developers can work on individual components and send/receive mock messages in order to test their functionality in the larger application context.

5.3.4 Configure space

Once developers have implemented and tested all the components in a macroworld application (over the course of several hours, days or weeks), they need to begin “packaging” it for students and teachers to use it. There are two steps in this process and the first one is to configure the physical space and decide which resources (i.e. devices) are needed for a particular application. *RoomPlaces*, and in particular the RoomPlaces GUI (figure 13), are the tools used for this particular task. The process is outlined in the diagram in figure 14 and it involves positioning stationary resources and configuring mobile ones to broadcast their location changes.

5.3.5 Create channels and packages

Once the physical space of the classroom has been laid out, the last part of getting a macroworld application ready to run in a classroom is to assign which interfaces are going to be on which devices and to who. The process to do so involves a series of sub-steps depicted in figure 14.

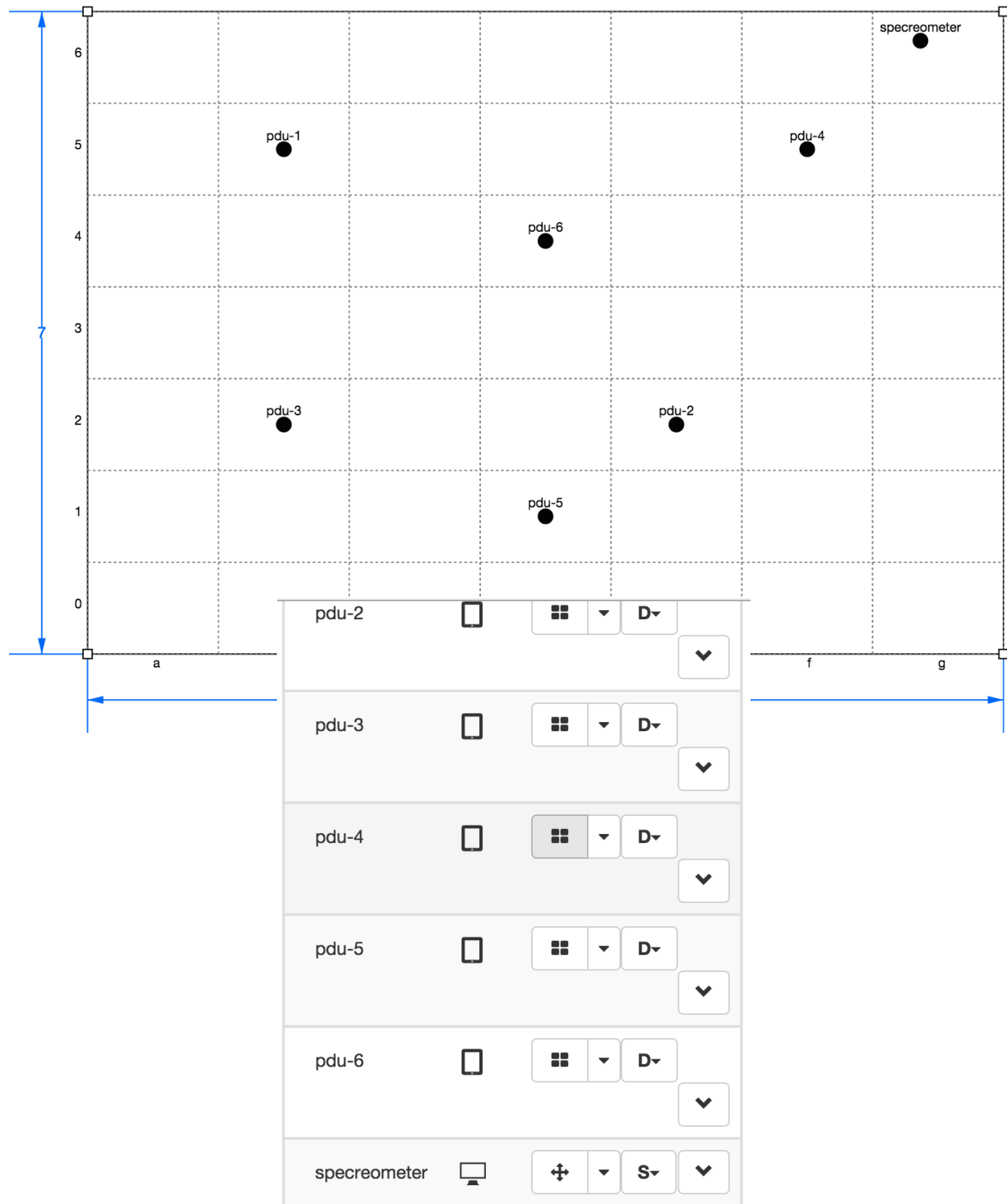


Figure 13. RoomPlaces interface. The RoomCast interface has two portions. The classroom map (top), where resources' location can be manipulated in real-time and the devices list and configuration view (bottom), where developers can configure parameters such as the type of resource, tracking mechanism and granularity. The map shows the layout of resources for AquaRoom.

This process involves creating sets of channels (i.e. interfaces), which I call packages, and then associating these packages to users and resources/devices. This

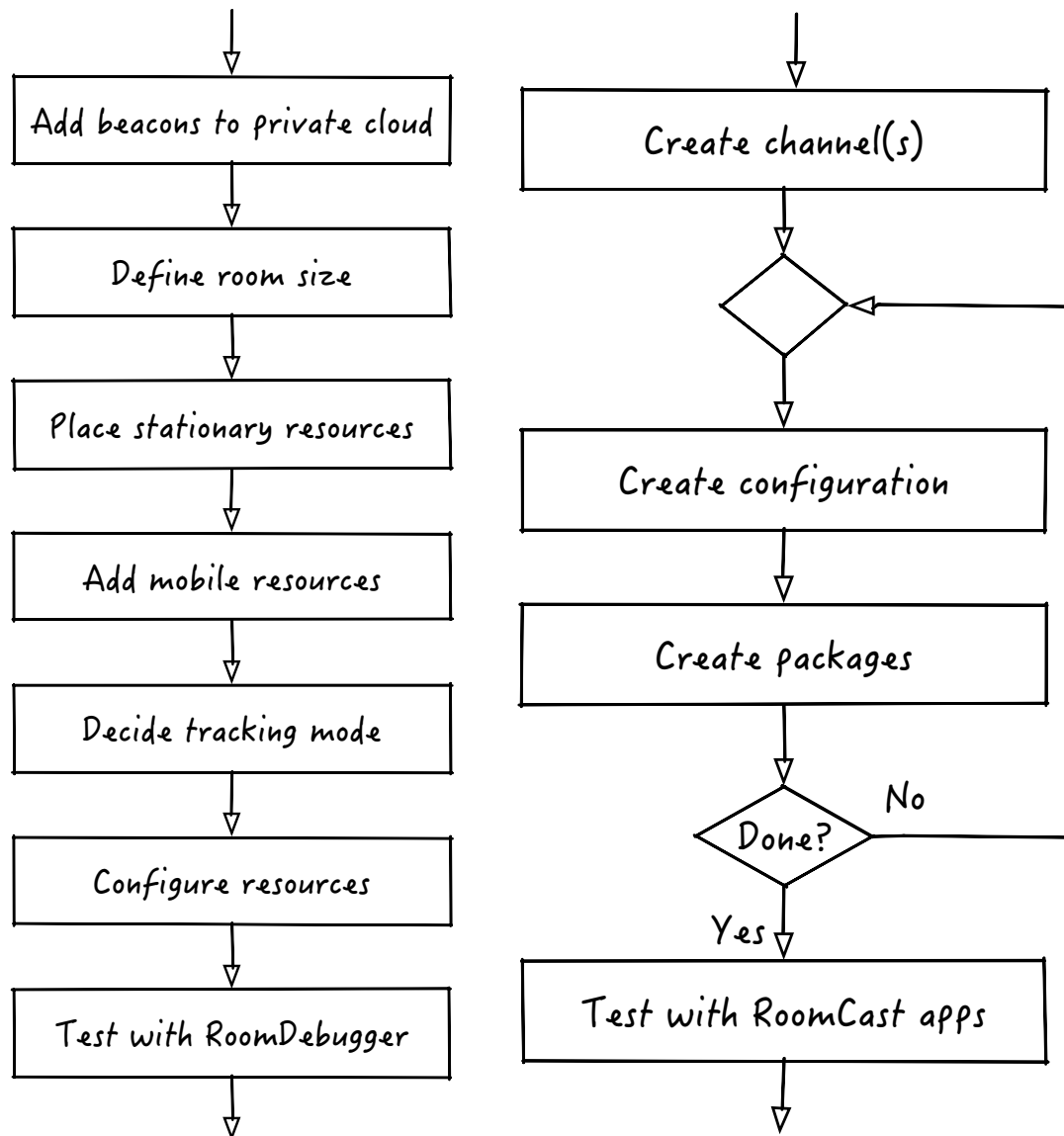


Figure 14. Steps involved in the configuration of the classroom space using RoomPlaces (left). This diagram is an "exploded view" of the box labeled "configure space" in the diagram in figure 10. Steps involved in the creation of RoomCast channels and packages (right). Performing these steps is necessary to "promote" interfaces to channels so that they can be used by students and teachers in classrooms.

decoupling of device, ownership of the device and content (which I call “late binding” of content to devices and users) provides developers (and teachers) with a high degree of freedom in terms of which content is accessed where. The ability of choosing at the “last-second” which interface goes to a particular display is a critical capability for developers and teachers to react to the volatility of the classroom environment, control it and provide the capabilities outlined in chapter 3. Once these associations (called configurations) are established, teachers can use a “classroom remote” tablet application to switch between them and control the classroom.

5.3.6 Testing and classroom(s) deployment

Once a new macroworld application is ready to be deployed in real classrooms, developers typically perform one or more complete system tests to verify its functionality. Such tests are often performed in the laboratory where the application is being developed and, after the setup has been completed at schools, a final smoke test is performed in-situ to ensure the absence of basic problems that will prevent the macroworld from working at all. In order to perform both laboratory testing and use in classroom, a macroworld application needs to be deployed first. Developers can do so by uploading it to a server (typically by cloning the code repository where the application is) and starting it using the framework core. nutella is capable of starting multiple instances of a single application, in order to support development/production

setups and enable testing. The framework core also takes care of starting RoomRecorder. This distributed logging system stores all messages that are exchanged across all components in the macroworld. Developers can also write to the distributed log explicitly using the nutella native-language libraries. This becomes useful whenever students' interactions need to be analyzed for research purposes or during testing to discover the root cause of bugs.

While macroworld applications are running in the classroom, developers have available to them RoomMonitor a GUI utility that allows them to monitor all components in all applications running on a certain instance of nutella.

5.3.7 Data analysis

Once the classroom enactment is complete (or even while it is in progress sometimes), researchers need to analyze the distributed application logs in order to gather evidence supporting their research. In order to do so they can use RoomRecorder APIs to filter and export data in different formats (JSON, CSV). I plan to develop a GUI in the future to facilitate this process and allow researchers to “replay” arbitrary sections of the logs in order to facilitate the complex logs sense-making process.

Chapter 6

Constructing and enacting macroworlds with nutella

In this chapter, I will describe how nutella can be used to build macroworld applications. In particular, for each capability identified in chapter 3, I will show how nutella implements it and how the framework features can be used to overcome many of the challenges that arise during the construction and enactment of macroworlds. All examples in this chapter use JavaScript / node.js.

6.1 Support multiple macroworld simulations types

nutella's support of simulated, emulated and generated phenomena hinges mainly on two features of the framework: the loose coupling between user interfaces and bots and the use of RoomComponents. More specifically, in nutella, simulated phenomena are implemented by a bot continuously updating the state of the simulation and by a series of interfaces displaying portions of the simulation state in the classroom and allowing students to manipulate it. Generated phenomena are similar, but students use their movement as an input to the simulation. Emulated phenomena instead are implemented with a bot that filters and adapts the data coming from the monitoring of a remote phenomena (or historical data) and a series of user interfaces displaying "slices" of such data.

The decoupling between user interfaces and bots provided by nutella plays an important role because, from an interface designer perspective, emulated and simulated phenomena are equivalent. Data comes to the interface and needs to be displayed, no matter where data comes from. The same is true for bots designers when looking at simulated and generated phenomena bots. It doesn't matter where the inputs to the simulations come from (an interface, human movement, tangibles, etc.) they all look the same to the simulation bot. These "equivalences" are important because they help reduce the viability that developers need to face and contribute to simplifying their job.

While the decoupling plays a role in reducing the variability between the three classes of macroworlds, RoomComponents play a similar role within each macroworld class. Consider AquaRoom and RoomQuake. They are both simulated phenomena, but their implementation of the simulation bot varies dramatically between the two. In AquaRoom, developers need to constantly update the simulation based on students' dyes injections. The simulation is interactive and the code will look something like this:

```
while (true) {  
    processInput();  
    update();  
    produceOutput();  
}
```

In RoomQuake, instead, after the quakes schedule for the whole unit is decided, no action is needed by the bot other than storing such schedule and sending it to the user interfaces that request it.

These two simulated phenomena are examples of two different sub-classes of simulated phenomena that I call *active* and *passive* phenomena. The way nutella provides scaffolding for both is via RoomComponents. By providing templates (i.e. RoomComponents) for both active and passive phenomena (in several programming languages) developers that want to implement a new active phenomena bot can install the appropriate RoomComponent in their macroworld application and then modify it to their needs. Any nutella developer can publish their templates to the nutella RoomComponents repository which enables the number of templates to grow with the community. More on this in section 6.6.

6.2 Support leveraging of the physical space of the classroom

RoomPlaces is the module in the nutella framework that is devoted to handling resources location and space. The module provides a series of tools to help developers deal with the challenges that emerge when leveraging the physical space of the classroom in macroworlds. In particular, the RoomPlaces focuses on supporting setting and getting devices' location within the classroom, at different granularities, independently from the devices location-sensing and communication capabilities. To enable this, this nutella module was designed around three central ideas: *abstract the different devices location sensing and network capabilities*, *define a unified model of location* and *provide a consistent interface*, across programming languages and devices, to access

and manipulate location and spatial information about the classroom. In order to demonstrate the capabilities of this module, I am going to present some examples.

First, there is the issue of dealing with resources location and different tracking and location reporting systems. Some macroworlds, such as Participatory Simulations and the Hunger Games, make use of *automatic tracking systems* that are able to detect proximity of kids between each other and the proximity of kids to particular areas of the classroom. Others, such as RoomQuake and HelioRoom, rely on manual measuring procedures and the transcription of these measures into some sort of configuration file or configuration Graphical User Interface. Finally, macroworlds such as AquaRoom, rely on students' self-reporting their location and transcribing such measures directly in the interface provided to them. These three ways of specifying the location of a single resource (automatic tracking, manual configuration, students' self reporting) are all handled by the nutella framework and, in particular, by the `nutella.location` APIs.

Before showing a real code example, I want to talk about a second issue that is closely intertwined to the first one, that is the fact that different macroworlds require different granularities of precision and coordinate systems when handling resources location. In RoomQuake, for instance, students have to identify the position of a quake with an (X, Y) coordinates pair on a 2D continuous space and the same is true for the position of the seismographs. In AquaRoom, instead, the self-reported position of sampling location is expressed in "battleship" coordinate pairs (e.g. A1, B4, G2) that are

discrete. Finally, in WallCology, location of WallScopes is simply expressed in terms of “hotspots” labeled with a name (e.g. Wall 1, Wall A...). Again, nutella provides a way of handling all these different *coordinate systems* and allows them to be used at the same time (with some resources using a continuous tracking system for instance and other using the hotspot). For instance, if I wanted to set the location of a RoomQuake seismograph in a continuous coordinate system in, I can simply do this with the following two lines of code:

```
nutella.location.resource['seismograph_1'].continuous.x = 5.3
nutella.location.resource['seismograph_1'].continuous.y = .2
```

Similarly, if I wanted to set the location of the portable drilling unit (where a student is logged in as ‘alessandro’) in AquaRoom, we could simply use the discrete coordinate system APIs like so:

```
nutella.location.resource['alessandro'].discrete.x = A
nutella.location.resource['alessandro'].discrete.y = 2
```

Finally, if we wanted to set the location of a particular WallScope in WallCology to be close to wall A I can do so by using the following API

```
nutella.location.resource['wallscope_1'].proximity.rid = 'wall_a'
```

As demonstrated by this three examples the APIs for continuous, discrete and proximity coordinate systems are roughly the same. What changes is simply the way

they are invoked. In the Hunger Games, the proximity APIs are called by the tracking systems which automatically detects the location of each kid, in RoomQuake and HelioRoom the continuous APIs are called by a GUI that is used to configure the room before the simulation starts (more on this later) and in AquaRoom, the discrete APIs are called by the portable drilling unit itself, whenever a kid uses it to self-report its location.

So far I talked about setting the location of a certain resource but not about retrieving it. This gives me the chance to talk about another feature of RoomPlaces, which is its ability to handle both resources that are both static and dynamic. In RoomQuake, for instance, seismographs never change their location across the unit (static resources) while, at the other end of the spectrum, kids are constantly moving in the Hunger Games (dynamic resources). The reason this is important is because the way the location of stationary and dynamic resources is accessed is different. Ideally, with slow or never changing information (such as the location of static resources), it would be better to be able to access it “as needed” while it would preferable to be notified whenever new information is available in a rapidly-changing information source (such as location in dynamic resources). Leveraging nutella’s native support for push and pull communication strategies, it is possible to “register” to receive location updates for an arbitrary set of dynamic resources like so:

```
nutella.location.resource['my_resources_group_id'].notifyUpdate = true
```

It is then possible to register a callback function that will be fired every time a location update is received. For stationary and mobile resources, it is also possible to “query” for their location as follows:

```
nutella.location.resource['rid'].proximity.rid
nutella.location.resource['rid'].proximity.continuous.x
nutella.location.resource['rid'].proximity.continuous.y
```

RoomPlaces’s design, decoupling the setting and retrieval of location for each resource, is the key to providing a homogeneous interface to locational information, independently of the technology used underneath. In addition to simplifying the development of applications today, this will allow developers to plug in different tracking technologies in the future that are not yet available today.

RoomPlaces APIs also allow developers to store arbitrary key-value pairs for each resource in the systems. This simplifies storage and retrieval of information that is logically tied to physical devices. In the Hunger Games, for instance, each kid had an iBeacon, stuffed inside a plush animal (i.e. their token), which they could use to forage. If a developer wanted to associate how many calories a certain squirrel had accumulated so far it could do so with the following line.

```
nutella.location.resource['my_squirrel'].parameter['tot_calories'] = 115
```

Again, the APIs to set key-value pairs are consistent across languages and allow developers to easily store and retrieve information tied to physical devices, even if these devices are incapable of storing information themselves, such as iBeacons. This enables a lot of scenarios where regular objects can be enriched with information in order to quickly design applications where the same physical objects have different roles. For instance, an iBeacon could be a squirrel in the Hunger Games while, at the same time, a different one could be used by teachers as a control object (to lock a shared display for instance).

Finally, as anticipated above, RoomPlaces provides a graphical user interface that “wraps” the APIs described above and can be used to configure and monitor the physical space of the classroom (figure 13).

6.3 Provide feedback during and after the enactment of macroworlds

nutella’s strategy to assist developers while creating user interfaces to provide feedback to students, teachers, and researchers hinges on nutella’s communication facilities, provided by the nutella native-language libraries and RoomRecorder. The first one provides support mostly for real-time, push-pull messaging while the second one provides support for batch, mostly a-posteriori aggregation. Together the two implement an information distribution mechanism that allows developers to access information wherever they need it and whenever they need it. This ability to freely

distribute information across the various components of the framework was one of the main design principles of nutella. For this the reason I like to call nutella an “information everywhere” framework. In the next paragraphs, I will demonstrate with some examples how this principle works in practice to assist developers in dealing with specific challenges they face while designing macroworlds.

6.3.1 Providing feedback for students and teachers

Despite providing support for making data available to geo-spatial and semio-spatial representations (and other types of user interfaces and visualizations used in macroworlds), nutella doesn’t explicitly provide support for actually creating such spatial representations. There are mainly two reasons for this. First, there is no reason to re-invent the wheel. There is already a plethora of excellent tools and graphic libraries (e.g. D3.js, Paper.js, Processing, etc.) to create a great variety of geo-spatial and semio-spatial representations. Second, providing students with feedback via aggregate representations of the data they collected so far requires representing application-specific representations that are dependent from the science domain the application is trying to support. A one-size-fits-all solutions for this problem doesn’t, unfortunately, exist. With these two constraints in mind, nutella has been created to afford interface designers an easy way to retrieve the data they need to display and to allow them to focus on what they do best: create the interfaces and design the interaction.

Consider take the aggregate representations of students' seismograph reading in RoomQuake discussed in chapter 3 (figure 2). The code necessary to retrieve the data that needs to be displayed by the interface, including the real-time updates, is the following (retrieved from <https://github.com/lgt-uic/roomquake/blob/master/interfaces/rq-aggregate-display/index.html#L203>):

```
var query_params = NUTELLA.parseURLParameters();
var nutella = NUTELLA.init(query_params.broker, query_params.app_id,
    query_params.run_id, NUTELLA.parseComponentId());

nutella.net.request("room_configuration", '', function(response) {
    configureRoom(response);
});

nutella.net.request("observations", '', function(response) {
    response.observations.forEach(function(e) {
        addObservation (e.seismograph, e.p_arrival_time,
            e.s_arrival_time, e.max_amplitude, e.sp_gap, e.distance,
            e.magnitude, e.quake_time);
    });
});

nutella.net.subscribe("new_observation", function(e) {
    addObservation (e.seismograph, e.p_arrival_time, e.s_arrival_time,
        e.max_amplitude, e.sp_gap, e.distance, e.magnitude,
        e.quake_time);
});

nutella.net.subscribe("new_quake", function(quake) {
    wipeObservations();
    setNextQuakeTime(quake.quakeTime);
});
```

After initializing nutella, developers request the room configuration (position of seismographs, size, etc.) and the observations that have been entered by students already. Then developers register listeners and callbacks for new observations (created by students using the data collection interface showed in figure 1) and the observations

wiping/quake change event that is generated by the teachers control panel. When a new message is delivered to the “wipe_observation” channel, the registered callback effectively providing developers with a mechanism to “push” data whenever it is available. Also, whenever a messages is received, nutella takes care of parsing it so that developers can access its content right away. This code should hopefully demonstrate how straightforward it is to distribute and retrieve information using nutella and how this could help interaction designers stay focused on the interaction design as opposed to shuttling information.

6.3.2 Providing preferential feedback to teachers

Suppose now I wanted to enhance RoomQuake with a notification system for teachers similar to the one created for WallCology and described in section 3.3.2 (figure 4). This system could compare the actual “quakes” generated by the simulation with the data entered by the students and notify the teacher if there was a discrepancy between the two. The logic for this system is easy enough that it could be implemented in the user interface directly but, to be more modular, it is better to move it to a dedicated bot. The code for this bot will look something like this:

```
// skipping nutella initialization

var currentQuake;

nutella.net.request('current_quake', '', function(response){
  currentQuake = response.currentQuake;
});
```

```

nutella.net.subscribe('new_quake', function(quake){
  currentQuake = quake;
});

nutella.net.subscribe("new_observation", function(obs) {
  var notifications = compareWithCurrentQuake(obs);
  nutella.net.publish('teacher_notifications', notifications);
});

```

Again, the previous examples should demonstrate how nutella helps developers focus on implementing the functionality without worrying about “low-level” details. The code should be self-explanatory since it is very similar to the previous example. The only difference is the use of `nutella.net.publish` to send notifications to the user interface which is waiting for them.

6.3.3 Providing a-posteriori feedback to researchers

Researchers typically access macroworlds applications logs in batch during or after macroworld enactments. Every time someone needs to access messages or logs in a-posteriori and batch way, they must use RoomRecorder. This module in the nutella architecture provides a set of APIs that allow any component in a macroworld application to retrieve data, according to a set of specified arbitrary filters. In order to understand how this works, consider the following example.

Suppose I was trying to create a web interface for researchers to download a snapshot of the data from yesterday's enactment of RoomQuake (researchers are only

interested in students' observations, not the rest of the messages and logs). Researchers need the data to be in CSV format so they can import and manipulate it in Microsoft Excel. The code to interact with RoomRecorder and retrieve such data is the following:

```
// Returns the CSV file URL
function getYesterdayObservationsCSV() {
  var now = moment();
  var yesterday = now.subtract(1, 'day');
  var filter = {
    channels: ['new_observation'],
    begin: yesterday,
    end: now
  };
  return nutella.net.filter(filter).asCSV();
}
```

After the function above executes developers will be able to take a URL and paste into a web page for researchers to access. If developers need more complex filtering (for instance over a series of intervals) or they are looking for specific patterns within a channel (e.g. new observations about seismograph 1) they can still use the `nutella.net.filter` function and simply pass different JSON patterns which will be used by the RoomRecorder framework bot to filter the messages and logs.

6.4 Support classroom orchestration during the enactment of macroworlds

Despite the several attempts at supporting teachers as conductor of performances described in chapter 3, more research is needed to better understand what kind of tools are needed to help teachers orchestrate macroworld classroom activities during their enactment. As a consequence, a one-size-fits-all approach won't work when trying to

support the construction and enactment of these tools. Nonetheless, nutella provides two main facilities that can be used by developers (and sometimes directly by teachers) to assist them in the creation of tools for macroworld classroom orchestration: the RoomCast macro-module and a way to sandbox macroworld applications built into the framework. The rest of this section presents four examples of the use of these tools.

6.4.1 Provide teachers with privileged access, control, and configuration over macroworld simulations

As said earlier, each macroworld simulation is different. Therefore, designing control interfaces for these simulations involves domain-specific knowledge that makes it impossible for nutella to provide complete support to their design. However, similarly to what happens for data-entry interfaces, nutella focuses on simplifying the process of exchanging and distributing information across all components of the application. nutella's *data everywhere* approach to information is used by developers designing simulations dashboards exactly in the same way it is used to design preferential feedback mechanisms for teachers.

6.4.2 Provide teachers with control over representational affordances, both public and private

In order to facilitate the teacher during the process of control of representational affordances, RoomCast provides them with a universal “remote” control for their classroom. This remote consists of an iPad app that allows teachers to switch between a set of pre-defined configurations. If I wanted to apply this to the RoomQuake example in section 3.4.2 the teacher interface will look like figure 15. This interface automatically reduces the size of each “configuration tile” to accommodate more configurations. In

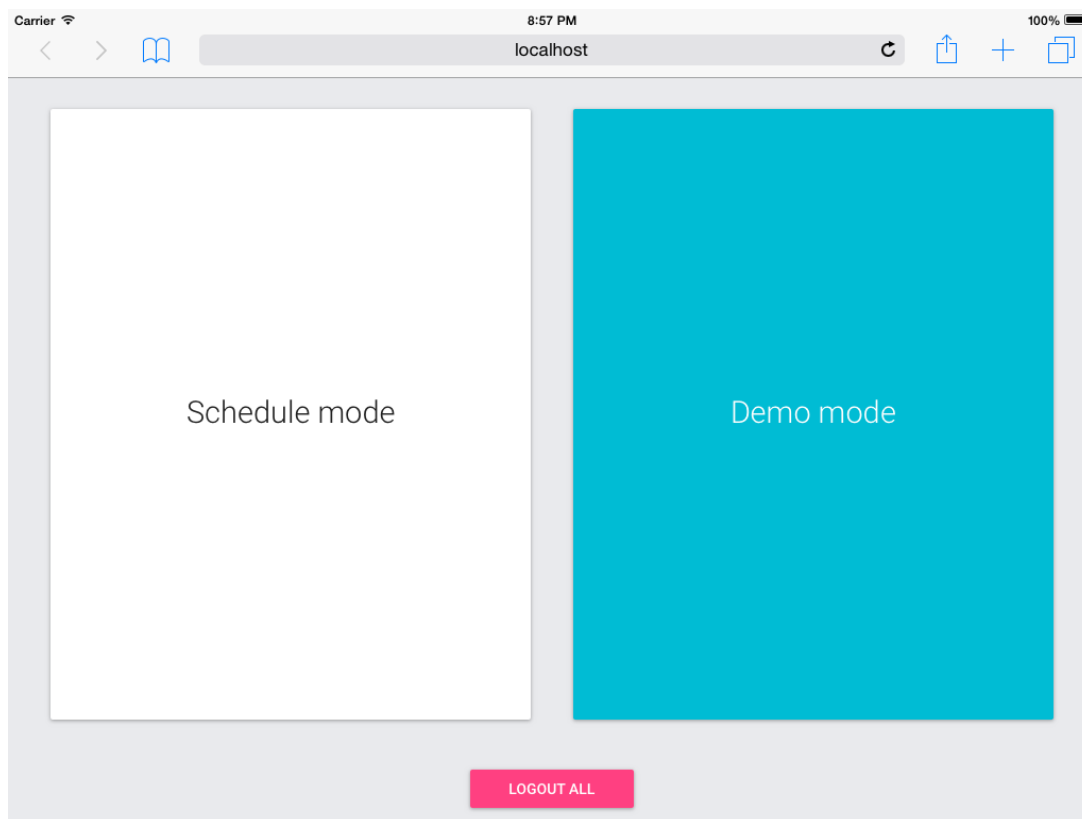


Figure 15. RoomCast “classroom remote” teacher interface. This iPad application provides teachers with a single control for all the macroworld devices and appliances in the classroom. In this picture, the remote is configured to switch between demo and schedule mode in RoomQuake.

order to switch to a certain configuration, the teacher needs to hold his or her finger on a configuration tile for three seconds. The rationale behind this choice is avoiding accidental switches. This app might seem extremely simple and that is exactly the point. Teachers enacting macroworld curriculum units rarely have time to look down at their iPads while teaching and managing all the variables that characterize a macroworld classroom. For this reason, I made the decision to leave the “online” interface provided to teachers during classroom enactments simple and offload the complexity to the “offline” configuration interfaces: the RoomPlaces *channel creator* and *packages creator* (more on these later).

Any component in the application can of course register callback that are triggered whenever there is configuration change.

```
nutella.cast.configurationChange(function(prev_config, new_config){
  // React to change of configuration
});
```

Components can also actively query RoomCast for the current configuration at any moment.

```
var cc = nutella.cast.currentConfiguration
// Use the configuration value, cc
```

Using this simple mechanism, teachers can change the state of all components in a macroworld with the single click of a button.

6.4.3 Provide a way to adapt macroworld applications to the technologies available in the classroom

In addition to providing teachers with control over the classroom affordances, RoomCast is also the way nutella supports different technology setups in different classrooms. RoomCast allows developers to associate interfaces, channels and packages to physical devices allowing them to re-configure this mapping at any time and, if necessary, in real time. The real-time component is especially important when dealing with the daily disruptions that characterize the classroom environment. A kid forgot their iPad, the projector in the room is broken, someone borrowed the laptop cart and didn't return it, all these are common occurrences in elementary school classrooms. RoomCast provides a way to re-configure where something is seen by who. The tools to do this are, as mentioned earlier, the RoomCast channel creator and package creator.

The RoomCast channel creator (figure 16) is used by developers to “promote” an interface that they created to be used by students and teachers in classrooms. Each channel is represented across all interfaces by a card with a name, an icon and a brief description. Once the channel has been created it becomes available in the RoomCast Package creator. RoomCast package creation is the interface responsible for creating channels packages and configurations (figure 17). Packages are simply collections of

channels identified by a name while configurations allow macroworld designers to store more than one association of channels to packages.

Using the RoomCast native applications teachers and students can log-in on any device in the room (iOS and OSX devices at the moment) and access the channels that are associated to their identity (or to the identity of a particular device, for ambient technology) at any time during the macroworld enactment. This decoupling of roles and devices, called *late binding*, enables the same interfaces to be put on any device in the room, provided that the interface was designed for that device and to account for devices with different screen sizes and capabilities.

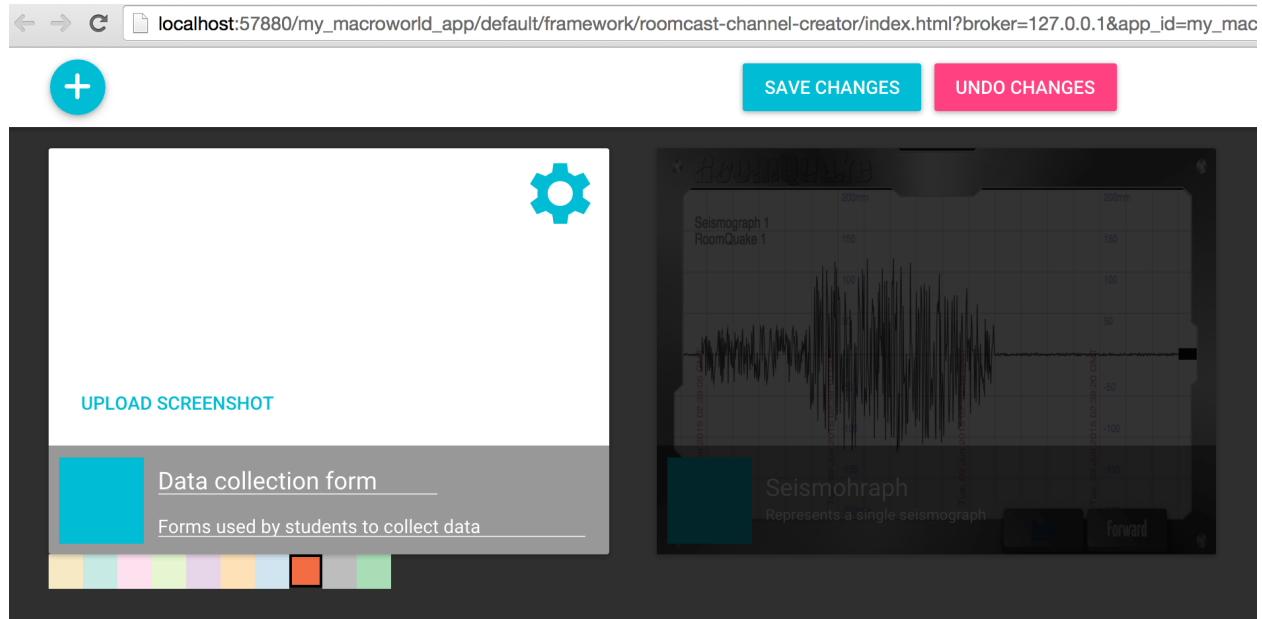


Figure 16. RoomCast channel creation interface (detail). Using this interface, developers can "promote" interfaces to channels so that they can be used by teachers and students. Each channel has a name, a description, a picture and a color icon that identify it. Channels are represented with the metaphor of a card.

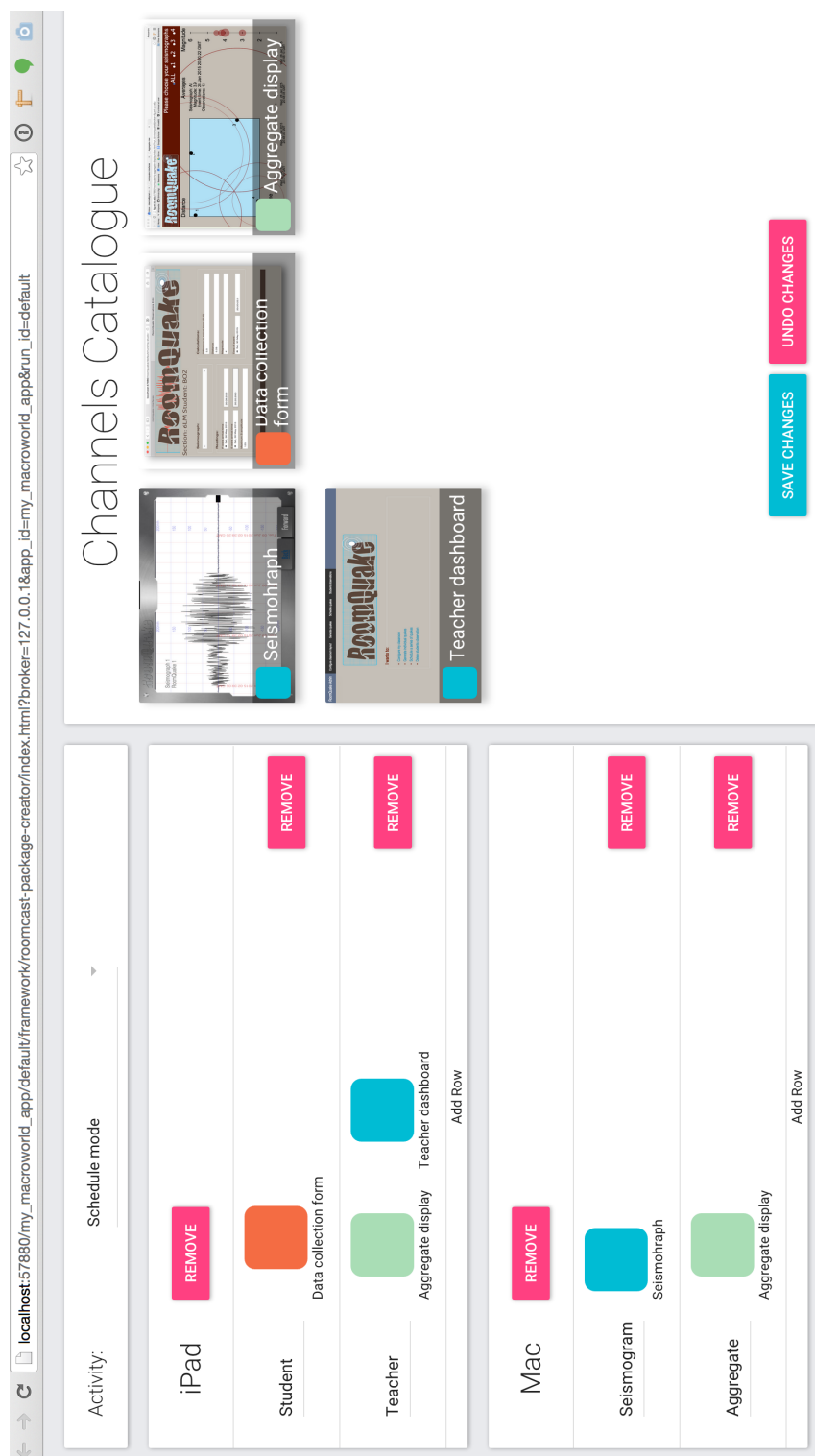


Figure 17. RoomCast package creation interface. This interface is used by developers create channel packages that are then associated to physical devices in the classroom via late binding. The interface has two main sections: a channel catalog and a packages configuration area.

6.4.4 Provide support for different instructional organizations

The way nutella provides support for different instructional organizations is by sandboxing applications and having built-in support to run multiple instances of the same application. nutella allows all the following scenarios:

1. a single teacher with multiple classrooms each in their own room (rotating teacher)
2. a single teacher with multiple classrooms, all sharing the same space but each working on their separate macroworld (rotating classrooms)
3. a single teacher with multiple classrooms, all sharing the same space and all working on the same macroworld (rotating classrooms)
4. a single teacher with multiple classrooms, all sharing the same space and portions of the macroworld simulation (rotating classrooms)

nutella achieves this because the notion of multiple, concurrent instances of the same application all running at the same time is built into the framework. In order to achieve this, nutella uses a data multiplexing mechanism to isolate communications and storage of data relative to a single instance of an application, called a *run*.

In order to enable developers to share components of a single macroworld application across different runs (allowing, for instance, multiple classrooms to work on the same phenomena but collect separate data) nutella has the notion of *application components*. These special components are shared among all the instances of a single

application escaping the data sandboxing mechanism used for regular components. In order to mark a component as an application component, a developer needs to modify the `nutella.json` file inside the `macroworld` application folder.

Here is how the previous four scenarios would be implemented in `nutella`:

1. Launch as many instances of the same application as there are classrooms.

Each instance will be isolated. User interfaces can run on different hardware or, if necessary, on the same hardware (i.e. sharing tablets or laptops between classrooms).

2. Launch as many instances of the same application as there are classrooms.

Each instance will be isolated. User interfaces run typically on the same hardware unless students have personal devices, which is also supported.

3. Launch a single instance of an application and all students will work on it.

4. Launch as many instances of the same application as there are classrooms and mark the components that are shared as application components so that they can be shared among classrooms.

Each application is completely isolated from the others so the same server is capable of serving multiple applications (and multiple instances of the same application) at the same time.

6.5 Support interoperability with other learning technologies

nutella's support for integration with other technologies begins with the choice of technologies and data formats. The choice of open source, ubiquitous technologies makes low-level integration statistically easier because no matter what system nutella is integrating with, the chance is high that the system will be using compatible technologies. At a higher level, nutella's strategy for integration is based on the fact that *"everything is a component"*. Any software loading the native-language nutella library can act as a nutella component. The only difference between these external components and the components native to a macroworld application is the fact that they can't benefit from the component lifecycle management offered by the nutella framework core. Apart from that, they can interact and benefit from all the APIs and functionalities provided by the nutella library described above.

6.6 Provide support for "non-functional" capabilities

nutella provides two separate tools to allow developers to diagnose and inspect components and macroworld applications: RoomDebugger and RoomMonitor.

RoomDebugger is simply a debugging, testing and components mocking tool that basically provides a Graphical User Interface for nutella.net methods (figure 18). Developers can use this interface to send and receive messages to any component in an

Room Debugger
Pub/Sub
Request/Response
RoomPlaces
my_macroworld_app
default

Publish/Subscribe debugger

Publish

Subscribe

Incoming messages

Channel	Message	From
test2	"hello on test2 channel"	{"type":"run","app_id":"my_macroworld_app","run_id":"default","component_id":"room-debugger"}

Figure 18. RoomDebugger interface (detail). This interface enables developers to inspect and mock components while developing new macroworld applications. The detail in the picture shows how it is possible to publish and subscribe to arbitrary channels.

application in order to debug and test communications. RoomDebugger is mostly intended as a tool to be used by developers during the creation of macroworlds.

The twin tool of RoomDebugger that should be used by developers during the enactment of macroworlds in classrooms is RoomMonitor (figure 19). This tool provides a graphical visualization of the health of all the components in a macroworld application. It allows macroworld application developers to subscribe to components failures with different levels of granularity (component, instance, application) and be notified in real-time via email. Using the very same interface developers can also drill down to the component level and investigate what type of failure occurred by looking at the messages exchanged by components and sending “probe” messages to the

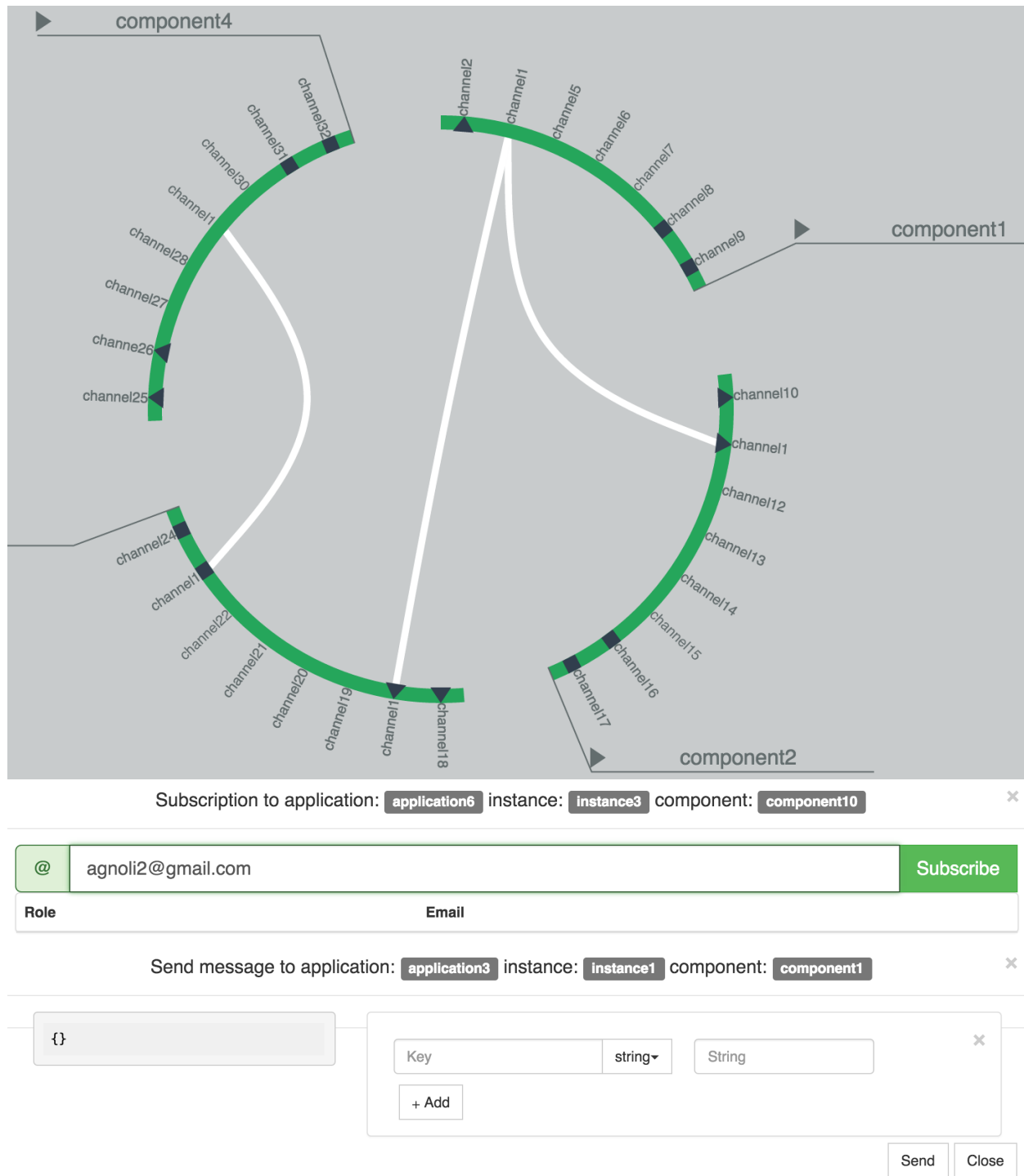


Figure 19. RoomMonitor interface (details). This interface enables developers to monitor macroworld applications during their enactment. The main view (top) provides a dynamic and interactive visualization of all the components within an application and the communication among each other (white lines). Developers can subscribe to failures at different granularities using a modal (middle) and can debug and troubleshoot by inspecting/sending messages to various components (bottom).

components or to other components.

In addition to these two tools to inspect and debug macroworld applications nutella is built with reuse and sharing in mind. As described earlier, components are the basic building blocks of macroworlds applications and nutella applications are simply collections of components stored inside a directory with a pre-defined structure and a `nutella.json` file describing some properties of the application. This organization of nutella applications in a pre-defined way has been inspired by a very popular web applications framework, Ruby on Rails, and it is a classic example of *convention over configuration*. According to O'Brien et al. (2010), this software design principle dictates that "Systems, libraries, and frameworks should assume reasonable defaults. Without requiring unnecessary configuration. Systems should 'just work'." The idea behind this software design principle is to reduce the amount of decisions a developer needs to make to "get going", allowing them to focus their cognitive skills on the task at hand instead of the minutiae of the implementation. In nutella, having a standard directory structure for a project makes it easier for developers to orient themselves and share entire applications knowing that other developers will know how to navigate them.

The use of convention over configuration in nutella doesn't stop at the application level: components are also organized in the same way. Since components are organized in a standard way, developers can share them with each other facilitating reuse. Moreover, since the standard to create components is also publicly available, any

developer can turn any piece of software that already exists into a nutella component and use it inside their application. To further simplify this sharing process I built a repository of components that can be installed simply referring to the component with their (unique) name. Besides the convenience of a quick installation process, the repository can be browsed and searched in order to simplify the discovery of components whenever they are needed.

Chapter 7

Early experiences with nutella

This chapter tells the story of how the nutella developers community grew over the past year. As developers were joining the community, they contributed their expertise and feedback and helped improve the design and implementation of nutella. The growth of the community can be divided into three main phases, each described by a section in this chapter. For each one of these phases, I will describe the state of nutella at that particular point in time, summarize the events that happened during the development phase and outline the lesson learned by interacting with developers engaged with nutella and the rest of the community.

7.1 Building RoomQuake with nutella

Despite the fact that it leverages work done over the past 6 years, the current version of nutella has been developed over the past year (June 2014, May 2015). In particular, during the first four months of development (June 2014, October 2012), I focused on implementing a working prototype of the system. During this time, I completed work on a first version of the nutella protocol, APIs, native-language libraries and the framework core.

During the first three weeks of November 2014 I, together with another developer and a designer, used nutella to develop RoomQuake, a 12-week-long curriculum unit on seismology and earthquakes. RoomQuake was the first, fully working, nutella application. It was composed of 5 interfaces and 2 bots which, taken together, implemented the macroworld application. In the previous chapters, I already described the students' data collection interface (figure 1), the interface used to visualize and interact with the aggregate representation of these observations (figure 2) and the interface used to control and orchestrate the whole unit (figure 6, 7). In addition to these three interfaces, a simulated seismograph interface was designed and implemented (figure 20) together with a score calculator interface used to determine how close students were in their estimates of the epicenter, magnitude and time of a certain quake.

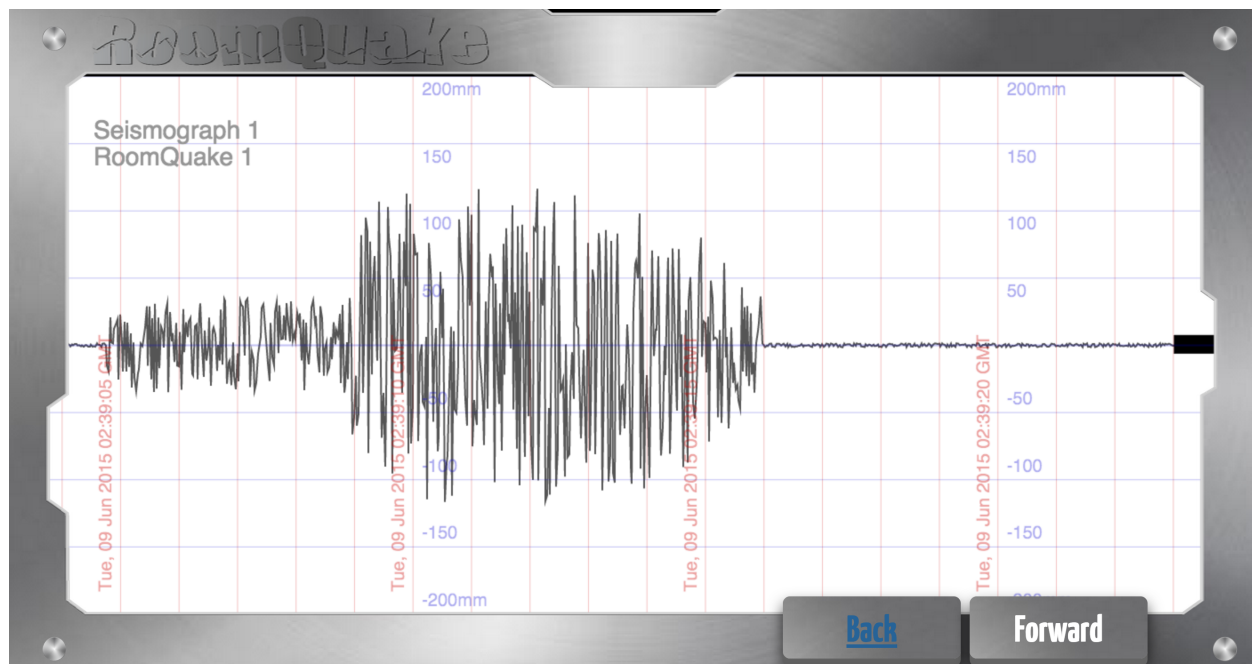


Figure 20. RoomQuake seismograph interface.

The backend portion of RoomQuake was implemented by two separate bots. The first one was dedicated to handling the simulation portion of the macroworlds (i.e. storing and managing the quakes schedule) while the second one was dedicated to storing and accessing student's observations.

7.1.1 Lesson learned

RoomQuake was enacted in three fifth-grade classrooms over a period of 12 weeks. During this time, nutella proved extremely reliable from a connectivity, communication and uptime point of view. This confirmed the reliability of the technologies nutella is based upon and the effectiveness of the nutella API layer design. Moreover, three people were able to collaborate on the same application and each work on separate components which were then integrated almost without change. This gave me confidence that the underlying design decision to organize macroworld applications as sets of components was helping collaboration.

Notably, the version of nutella used to build and enact RoomQuake didn't have any of the high-level framework components and macro-modules (i.e. RoomCast, RoomPlaces...) yet. This was addressed in the following development phase.

7.2 Building nutella's macro-modules

In December 2014, two developers (computer science graduate students) joined the nutella development team and began work on two of its macro-modules: RoomPlaces and RoomCast. They worked on nutella for 6 months (Dec 2014-May 2015). In January, nutella was also used in the CS 422, User Interface Design class by a group of three students (including one of the two developers mentioned above). The group contributed another component to the framework (RoomMonitor) over the course of four months (January 2015, April 2015). The nutella development team completed the design of all higher-level framework components during first four months of 2015.

During this second phase of development, as a team, we adopted an agile software development methodology called SCRUM (Schwaber, 2004). According to this methodology, the development of a new software system is broken down into a series of two to four week-long iterations called *sprints* (two weeks-long in the case of nutella). Each sprint is a time-boxed development effort that aims at implementing a restricted set of features and deliver, at the end of the iteration, a working product that could potentially be shipped. The methodology is particularly indicated to guide the development of software with fast-changing or ill-defined requirements.

Two of the staples of SCRUM are the *sprint retrospective meeting* and the *project retrospective meeting* (or *post-mortem meeting*) which happen at the end of each sprint and at the end of the project respectively. Both these events provide the opportunity for the

development team to “inspect itself” and create a plan for improvements to be enacted during the next sprint or project. During these meetings, participants often have, readily available, the products of their work and the project documentation (such as the finished product, code commits, code documentation, emails, meeting notes, etc.) in order to assist their recollection (Wohlin et al., 2003).

While working on high-level components the team “talked” to the language specific APIs and the lower level components of the framework a lot, strengthening and refining their design. In particular, the issues that were identified during a spring would be discussed during the sprint retrospective meetings and a plan to address them was drafted accordingly.

7.2.1 Lesson learned

One of the best examples of improvements in the low-level nutella APIs triggered by an unexpected need that arose while developing higher-level components is what we called framework-run communication. nutella’s communication APIs were originally developed to support self-contained applications and to enable its components to exchange information as described in the previous chapter. In this model, messages can be addressed to components by simply calling them by name. However, with the introduction of framework level components, the need for a different way of addressing components arose. The problem was that, for framework level

components, span across multiple applications and, therefore, they need to have a mechanism that enables them to select a particular component in a particular application and not simply a component.

This, in conjunction with application-level components, created a complex three-layer communication system that underwent complete re-design in order to make it approachable for end-users of the framework. Our solution was to use an hierarchical organization of these “communication” levels so that only framework-level components can explicitly address run-level components (i.e. the components used normally by developers of macroworlds applications). This way we were able to encapsulate the complexity at level of the framework that are only accessible to framework maintainers.

7.3 Building macroworlds with nutella during a three-day hackathon

In mid-May 2015 the framework reached a level of stability that allowed its development team (5 people at this point) to open it up to a selected group of 3 developers for an external evaluation. The evaluation of software by its users has a long tradition in the discipline of software engineering. As suggested by Davis (1995), the best way to assess what software users really need is to give them a “working system” and let them carry out “authentic tasks” using the system. However, as pointed out by the author himself, this methodology is often impractical due to the demanding

requirements on the amount of resources and time it imposes. This methodology is particularly challenging if the system is still in the early stages of development.

To circumvent the limitations of the methodology outlined above, software engineers have long used software prototyping (Brooks, 1975) and iterative software development (Larman & Basili, 2003) as a way of evaluation of software in its early stages. More recently, with the software engineering community embracing qualitative methods for their empirical studies (Seaman, 1999), hackathons have become significantly more popular, especially in the industry, as a way of collecting user feedback during the early phases of development of frameworks (e.g. Schreiner et al., 2015) and software systems in general (Raatikainen et al., 2013).

A hackathon (a portmanteau of the words "hack" and "marathon") refers to an event where small groups of developers (and sometimes UX designers, project managers and other involved in software development) participate in an intensive software prototyping activity for a limited amount of time (typically from a day up to a week). The goal of a hackathon varies, but hackathons are often organized around a specific topic, such as the programming language used by the participants, a particular software framework, API or type of application developed.

The structure of hackathons varies, but this type of events usually starts with some presentations introducing the technologies and the event itself. Then participants propose ideas or choose from a pool of available ideas and organize into teams. The

main phase of the hackathon involves teams rapidly prototyping their ideas. The outcome of this phase is typically working software that is demonstrated to other participants (and sometimes spectators) at the end of the event. Often hackathons involve a competition element where a panel of judges select the winning teams and awards them a prize.

Hackathons are appealing as a way of evaluating software by its users because they provide a “realistic, efficient, and effective means of holistically testing the ecosystem including technical details but especially the overall design and developer experience” (Raatikainen et al., 2013) and a way to perform what Jackson et al. (2013) call tutorial-based assessment, a technique used to provide a pragmatic evaluation of usability of software “as is”. Moreover, hackathons proved to be a good trade-off between the amount of resources needed and the authenticity of the evaluation where small teams get to build working prototypes while at the same time collaborating and learning from each other as a community of practice (Lave & Wenger 1991). People get to build the full system and, therefore, exert the whole framework. They have to work collectively and support each other like a real community.

As described earlier, developing and enacting a macroworld application is a complex process that involves a number of actors in addition to developers (such as researchers, designers, teachers, students) and lasts typically a few months. Similarly to the larger software engineering community, I reached the conclusion that evaluating

nutella by closely following a group of professional developers using the framework to build and enact a or more real macroworld applications is not appropriate at this stage of the development process. Therefore, in order to gather feedback on the viability of nutella as a framework to build and enact macroworlds, I turned my attention hackathons.

7.3.1 nutella hackathon

The event itself was organized into three separate days. During the first day, the facilitator helped the participants install the framework on their machines and provided training to teach them how to use nutella to build a macroworld. During the second day participants selected from a set of proposed macroworld applications ideas, organized into teams and began building their application. The third day was devoted to more coding and building and, at the end of the day, teams presented and enacted their application for other participants acting as teachers and students.

Participants were selected based on set of pre-requisites:

- Participants needed to bring a laptop with OS X or Ubuntu installed on them, a text editor and make sure they had internet connectivity.
- At the moment, nutella supports only JavaScript and Swift as interface languages and Ruby and JavaScript/node.js as bots languages. Participants needed to be familiar with at least one interface and one bot language.

- Familiarity with the Unix shell, git, and Github definitely helps was strongly encouraged as well as a little knowledge of the Ruby programming language and its ecosystem (i.e. gem, RVM).

A total of three participants attended the hackathon. All the participants are seasoned developers with four years of experience designing and creating learning technologies applications. They all had been exposed to macroworlds before and participated in projects where they needed to at least interact with a macroworld simulation. In particular, one of the participants had prior experience building macroworld simulations such as the Hunger Games and HelioRoom. In terms of their programming languages knowledge, the first two participants considered themselves JavaScript developers while the third considered himself equally proficient in JavaScript, Objective-C/Swift and Java.

7.3.1.1 Day one

The first activity of the first day was an “install-fest” where the facilitator helped the participants install nutella and all its pre-requisites on their laptops. At the end of this activity, all participants were able to type the nutella command in their shells and successfully see a welcome message. The goal of this activity was to smooth entry barriers into the framework and to “level the playing field” among the participants. Even with the right pre-requisite knowledge, while installing the framework many

potential roadblocks could arise, preventing participants from successfully participate in the hackathon. Being able to install nutella is not a measure of its viability its just a measure of its adaptability, how streamlined the installation process is and how complete and clear the documentation is.

The predicted length of this activity was a couple of hours, but one participant was able to complete the activity, mostly by himself, even before starting the hackathon. Another participant was able to complete the setup process, assisted by the facilitator, in 35 minutes while the last one, which started with a brand new laptop, was able to complete the setup process (including installing all the pre-requisite software) in a little under an hour.

After lunch, the facilitator guided the participants through and interactive training session (i.e. tutorial) that took them from an empty macroworld application to a fully working application, created using nutella. Training started by recapping what macroworld applications are and the typical process that drives their development and enactment. After the facilitator gave this brief introduction, it allowed each participant to proceed at their own pace following the online tutorial (https://github.com/nutella-framework/docs/blob/master/getting_started/tutorial_1.md) and inviting participants to ask questions whenever they needed help or clarification. The predicted length of this activity was the whole afternoon and the estimate proved to be accurate.

At the end of the day, facilitator and participants engaged in a 30 minutes discussion aimed at gaining feedback and first impressions from the participants.

7.3.1.2 Day 2

The day started with the facilitator dividing the participants into two teams, a two-participants and a single participant one, and distributing to each one of them an “initial design” for a macroworld application. The two-people team was assigned to the implementation of a simplified version of AquaRoom. Here is the “initial design” specification that they received:

AquaRoom is a simulated macroworld where students take on the role of hydrogeologists with the task of mapping a subterranean aquifer system (mapped to their classroom floor plan) and to use this information to decide where to locate a new chemical plant within the “local community” to minimize potential environmental impacts. In order to accomplish this task, students “inject tracer dyes” and “obtain water samples” using a portable tablet-based “drilling unit.” Students can only inject dyes and take samples at certain locations in their classrooms where there are “wells” that provide access to the underlying aquifers. Wells are strategically arranged in a grid pattern which can be indexed using the tiles on the room’s drop ceiling. Test tubes, enhanced with iBeacons serve as simulated dye sources and sample repositories. Students “inject” dyes by walking with their tablet drilling unit and dye-filled test tube to a well and using the tablet interface to confirm their injection. “Water samples” are subsequently collected in a similar fashion, and tested for the presence of dyes using a simulated spectrometer represented by a shared desktop computer. To analyze a sample, students walk to the simulated spectrometer with the sample, put their sample in the “analysis chamber”, click analyze and read the results of their sampling on the screen. The injection of a dye followed by sampling allows students to establish the presence of an aquifer, the direction and rate of flow, which are marked on a collective classroom map.

The one-person team was assigned to the implementation of an emulated macroworld that replayed pictures of animals captured using camera-trap technology.

Here is the “initial design” specification the team received:

In ClassroomSafari students are tasked with studying the wildlife of the savannah co-occupying the physical space of their classrooms. Students do so by placing food patches and “camera traps” at specific locations in their classroom. These virtual camera traps work like their real-life counterparts and take a burst of pictures (usually with 1 second interval between them) whenever “motion” is detected. When a virtual creature forages at a certain patch, the camera trap at that patch is activated. Camera traps are represented by ambient displays (i.e. iMacs, laptops) that display the pictures in real-time, as soon as they are taken. The interesting part of this macroworld is that the pictures displayed by the ambient affordances in the classroom are part of a data set of real camera-trap pictures taken, over the course of three weeks, in the context of an actual population-ecology study in Kenya. In addition to the simulated camera-trap interface, ClassroomSafari provides a dashboard interface where it is possible for the teacher to view the whole data set on a time line and choose which portion of it to “replay” in their classrooms. Teachers can choose arbitrary time intervals in the original pictures sequence and replay them, compressed or extended, in their classroom. Intervals between pictures are automatically adjusted, maintaining the correct proportions among them, by the ClassroomSafari simulation engine.

After receiving their “assignments,” participants were encouraged to ask clarifying questions. Developers in both groups were also encouraged not to worry about implementing the whole design but to proceed from the most to the least critical functionalities, based on their judgment. Participants spent the rest of the day working on their macroworld application prototypes. Similarly to day one, the facilitator invited participants to proceed independently and ask whenever they were stuck, had

questions or needed clarification. The facilitators also encouraged participants to help each other out, sharing their expertise and their work. These conditions were meant to simulate the working environment participants are used to operate in: small teams working on independent projects but often collaborating with each other.

7.3.1.3 Day 3

The majority of the third day was dedicated to allowing participants complete the coding of their macroworld prototype. At the end of the day, each of the two teams did a short demo of their macroworld application to other participants acting as “teachers” and “students”. At the end of this activity, the facilitator and the participants engaged in an hour-long discussion where participants were asked, once again, to share their opinion and feedback on the framework.

7.3.3 Results

All the participants in the hackathon were able to successfully use nutella to create a macroworld application prototype. As described earlier, the development and enactment processes of a macroworld application can be broken down into a series of steps. These steps, and the tools that nutella provides to support them were covered in the day-one tutorial and subsequently applied by developers while creating their macroworld application, as demonstrated by the artifacts that they produced. All

developers were able to move their application through all the steps that characterize the lifecycle of a nutella application (figure 10).

The AquaRoom prototype macroworld application contained two interfaces (figure 21) and one bot. Interfaces were simple web GUIs but they implemented the basic functionality of dye injection / sampling (portable drilling unit interface) and sample analysis (spectrometer). The ClassroomSafari application instead was composed of one bot and one interface. The bot was in charge of replaying, over a specified time interval, a set of camera traps pictures while the interface was in charge of displaying the images sent to it.

7.3.3.1 Support multiple macroworld simulations types

During the hackathon participants implemented two different macroworld simulations types: a simulated (AquaRoom) and an emulated (ClassroomSafari) one. Developers in each team used the same RoomComponent (`active-simulation-js-bot`) to scaffold the development of bots in both macroworlds. The ClassroomSafari group used the event loop provided by the bot in order to constantly look for pictures to send and actually send them to the appropriate display at the right time. Developers of the AquaRoom prototype didn't use the event loop because they didn't have enough time to implement the logic that models the water flow in the aquifers underneath the classroom since they focused on different elements of the macroworld simulation.

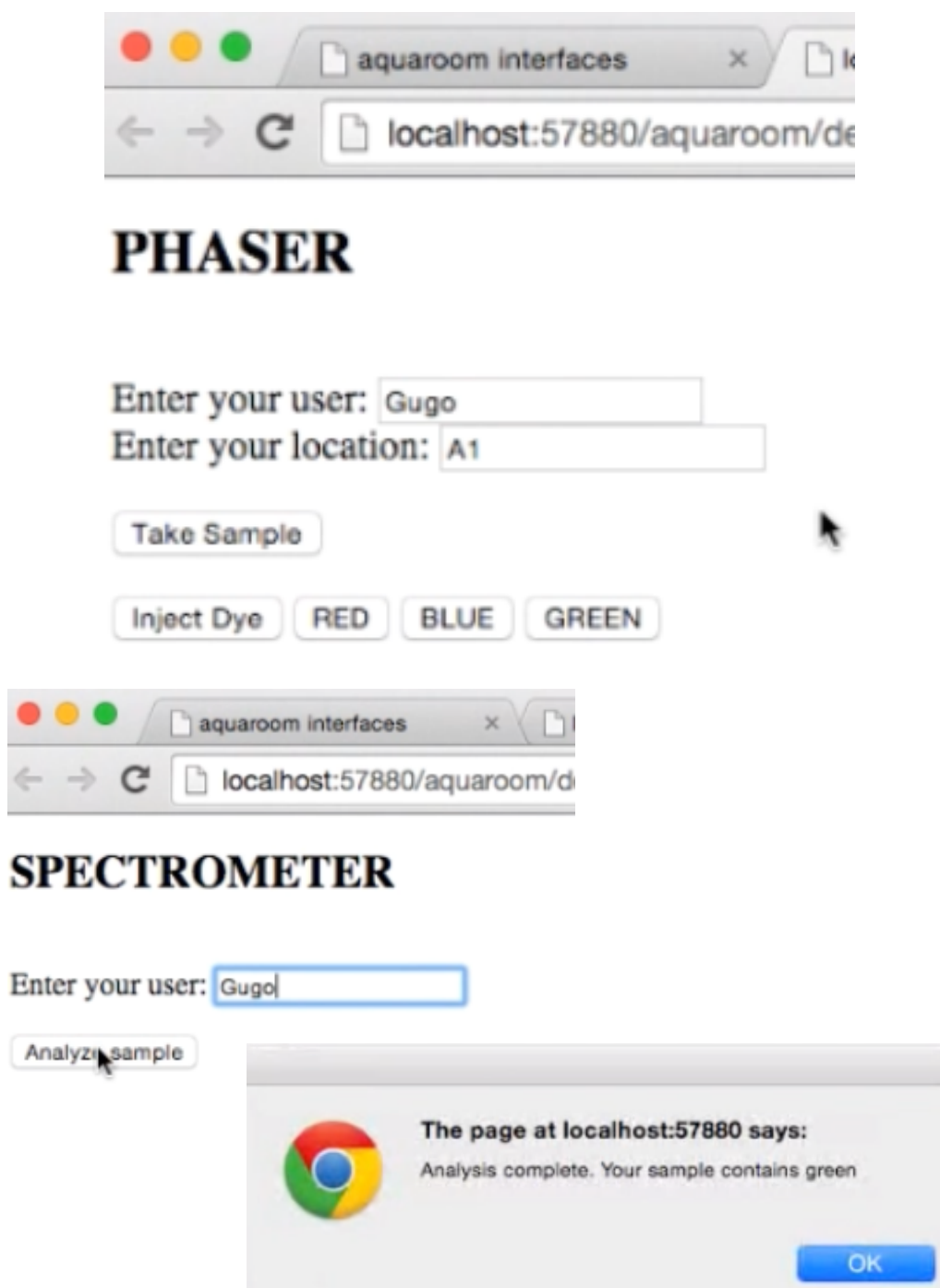


Figure 21. AquaRoom prototype built by hackathon participants (details). Two hackathon participants were able to build a simple AquaRoom prototype with two interfaces and a bot. Using the first interface (top) developers could inject dyes into the simulated aquifer and take samples. Using the second interface (bottom) they could perform an analysis of the water samples and confirm or reject the presence of a dye.

7.3.3.2 Support the leveraging of the physical space of the classroom

Both macroworld prototypes developed during the hackathon made intensive use of RoomPlaces. AquaRoom developers leveraged RoomPlace's discrete tracking mode to create the grid of wells used by kids to perform injections and samplings. For each tile in the grid, developers positioned an iBeacon into it to enable the portable drilling unit to detect which well the students are working on.

The ClassroomSafari group used RoomPlaces to define the location of virtual camera traps and food patches. After some discussion with the facilitator, the group decided to use RoomPlaces's hotspot tracking mode to assign a unique identifier to each camera trap that was then subsequently used in the simulation code to uniquely identify each ambient display.

7.3.3.3 Provide feedback during and after the enactment of macroworlds

During the hackathon, due to the restricted amount of time available to them, participants focused on the design and implementation of the "scientific phenomena simulation" portion of their macroworld prototypes. As a consequence, neither group created a data entry interface for students and/or an aggregate interface to provide feedback to them. However, the communication and "data everywhere" APIs used to build this kind of interfaces are exactly the same that are used to build the simulation portion of the macroworld prototype. Therefore, even if the communication APIs

weren't explicitly leveraged to build data collection and feedback interfaces, they were still leveraged to build simulation interfaces and visualizations. Particularly interesting is the fact that ClassroomSafari made extensive use of nutella's "binary communication" capabilities which provide a set of simplified APIs that facilitate the exchange of non-textual messages between components.

Moreover, developers experimented briefly with the logging and filtering APIs at the end of the demo enactment on day 3. They collectively tried to create a report for an imaginary researcher interested in looking at AquaRoom data and, in particular, at patterns in dye injections and sample taking. Participants achieved this by accessing the collection of all messages automatically stored by RoomRecorder and filtering this collection by channel using the appropriate nutella APIs.

7.3.3.4 Support classroom orchestration during the enactment of macroworlds

Similarly to what has been reported in the previous section, there was no opportunity to leverage all the capabilities provided by nutella in this area. However, all the participants were able to run more than one instance of their macroworld prototypes, simulating a scenario where multiple classes are using separate instances of the same application at the same time.

There was also no opportunity to leverage the RoomCast "room remote" iPad app due to some bugs and code signing issues. However, participants successfully used

RoomCast to create channels for the interfaces they designed and then to assemble those channels into packages. Developers were also able to exercise some “late binding capabilities” by running the interfaces they created on each others laptop and by accessing such interfaces also with their phones. Since none of the teams decided to create native iOS applications, participants didn’t have the opportunity to test RoomCast’s support for this type of interfaces.

7.3.3.5 Support interoperability with other learning technologies

Even if developers didn’t have the opportunity to make their macroworld prototypes interact with an external application, they demonstrated to understand the “everything is a component” strategy behind it as demonstrated by the following verbal exchange during the hackathon.

Participant: So all I have to do is import the nutella library, configure nutella, create a nutella variable with it and boom I can use it. Then my app is just a nutella component. Oh that’s brilliant! Oh but I guess then I’ll have to run it myself right?

Facilitator: Yes, that is correct.

Participant: Ah but that’s good because that way I can run it in my server and the app can talk to the broker on your side. Ok, that makes sense.

7.3.3.6 Provide support for “non-functional” capabilities

RoomDebugger was heavily used by all hackathon participants and praised as one of the best features of the whole framework since it allowed developers to mock

components, inspect and debug issues with inter-components communications. Participants also demonstrated interest for RoomMonitor but, besides looking at the interface briefly during their own AquaRoom demo, they didn't have the opportunity to test the full set of features that RoomMonitor offers to developers. Participants also commented that, even if the interface seemed to be very promising and useful, it was hard to evaluate it over a short period of time and since its best use case is for long running, real-world macroworld enactments when, sometimes, components crash for unexpected reasons.

7.3.4 Lesson learned

Through the whole duration of the hackathon, participants maintained a positive opinion of nutella. In particular, developers were impressed by the framework's communication facilities and its ease of use. No major flaws were pointed out by the participants although they discovered and highlighted a number of minor bugs. Often, participants created Github "tickets" on the spot for the bugs they discovered and some of those issues have already been addressed by the framework developers.

The most interesting outcome of the hackathon, at least from the point of view of the framework designers, is the fact that participants pointed out a series of concerns that they would like to see addressed, together with some features that they thought were missing in nutella. In particular, three main concerns emerged.

Participant felt that the persistence APIs provided by nutella weren't as polished and functional as the communication APIs, or other portions of the framework. As pointed out by one of the participants:

“we had some trouble with syncing up our projects over Github - again, this was mostly a data storage issue, and not too hard to solve”

These issues with the persistence layer were caused by a rushed design decision of the framework author. nutella has been designed to keep each separate instance of the framework self-contained. The reason for this was to provide developers with a way of keeping a developing environment on their machines and a production environment on some publicly accessible, remote server. Unfortunately, the consequence of this decisions is that data is bound to nutella and not to the application itself, which caused synchronization problems when two developers tried to collaborate on the same application. This issue didn't emerge while working on RoomQuake because developers were collaborating directly on the production instance of nutella, which caused this issue to remain latent. Fortunately, this can be easily addressed by allowing developers to specify where a certain macroworld application should store data.

However, this is not possible at the moment in nutella and, as pointed out by hackathon participants, this should be also address. nutella developers should address this and other scalability and security concerns. In particular, right now nutella relies on a single broker and database, which must reside on the same host. Providing a way for the database and the broker to be separate and their access, password protected or

somehow authenticated are necessary next steps to make nutella available to a bigger community where such concerns are of primary importance.

Third, participants lamented how, sometimes, the “debugging” capabilities of nutella were not adequate to their needs. As pointed out by two developers during the hackathon

“Another point of problems might be the fact that nutella does certain things and it takes a bit of digging and learning to understand what they do and how to trouble shoot them. For example, sometimes a [framework] bot dies off and it is not right away visible that the bot died nor why the bot crashed.”

“Weakness: Notifications that your [framework] bot and or instance failed or stopped. Many times I ran into the problem of an unresponsive app that was caused by the server crashing.”

Both comments above refer to the fact that framework bots lacked a proper logging mechanism, frustrating developers’ attempt of debugging them. Again, this is the result of a superficial design decision of the framework author. The (incorrect) assumption that lead to such decision was the fact that framework bot should have been stable enough not to need a proper logging mechanism. Unfortunately, this proved false, and forced developers to use a cumbersome work-around to verify the framework bots were operating properly. Luckily, this issue is also easy to address by leveraging the distributed logging mechanism provided by nutella (RoomRecorded).

Chapter 8

Conclusion, limitations and future work

The work presented in this dissertation focused on macroworlds, a learning technology that provides engaging ways for students to “experience” and interact with classroom-sized simulations of scientific phenomena. In this context, this research identified five application-level affordances of macroworlds and the corresponding capabilities that they impose on technologies powering these learning environments. A software framework (called nutella) implementing the capabilities described earlier was designed and implemented. Finally, this dissertation reported on some early experiences of developers using and building the framework. The outcome of these experiences provides some early evidence to support nutella’s viability as a tool to support the construction and enactment of macroworld applications.

Despite being designed explicitly to support the construction and enactment of macroworlds, nutella could be adapted and reused in different domains and environments, such as at-home entertainment and work-related applications in office environments. The reason for this is that, at its core, nutella can be seen as a “distributed simulation engine”. Exactly like nutella can power scientific phenomena simulations it could as well power simulations of war rooms, cockpits and, in general, any applications that requires distributed cognition (Hutchins, 1995). Similarly, it is possible to envision scenarios where a scientific phenomena simulation is replaced by fiction or

narrative and the classroom space is replaced by a living room or even a full house. As one of the nutella contributors pointed out:

“Finally, even if the framework is specifically meant to be used in the context of the classroom and for learning technologies applications, I can foresee a much wider range of projects which could leverage the power and simplicity of nutella in many other domains.”

There are of course some limitations to the types of simulations and narratives that can be experienced with nutella. Some of these limitations are technical limitations that are “inherited” from the technologies nutella is based upon. As an example, the choice of MQTT as the main communication protocol, which is in turn implemented on top of TCP/IP and Websockets, imposes limitations on the latency of messages. This imposes limitations on the speed of synchronization of two components. In addition to technical limitations, the framework still assumes a certain degree of supervision by developers. For instance, if one of the bots crashes teachers need to rely on developers to bring it back up. Ideally, macroworld applications should be as easy to install in a classroom as app on an iPad but that is not possible with the current state of the framework. Again, this kind of issues is typical of software in its infancy and will certainly be addressed as the development progresses.

Moreover, as pointed in the previous paragraph and by several subjects during the hackathon, nutella is still under development and it needs polishing in several areas. In addition to bug fixing, and addressing the issues pointed out by the developers’ community, the nutella development team is already working to implement

new features to improve the accessibility and usefulness of the framework. One of the first things we are going to focus on is implementing nutella libraries in more languages in order to lower the entry-barrier for even more developers. Moreover, we would like to incorporate different tracking technologies (e.g. Microsoft Kinect) and increase our support for more native platforms and operating systems (e.g. Android, Windows). We would also like to simplify the framework installation process, the distribution of macroworld applications and provide better authoring tools so that students and teachers could, eventually, start editing and creating new macroworld applications.

From a research point of view, nutella is ready for a more formal evaluation “in the wild”. This will require the framework being used to create and enact a real macroworld application and enact it in real-world classrooms. We are working to make this happen in late-Summer/Fall 2015 since the Learning Technologies Group and EncoreLab will be working with the framework to implement a new iteration of the WallCology macroworld.

REFERENCES

- Aarts, E., Harwig, R., & Schuurmans, M. (2001). Chapter 'Ambient Intelligence'. *The Invisible Future: The Seamless Integration Of Technology Into Everyday Life*, McGraw-Hill Companies.
- Åkesson, K. P., Bullock, A., Rodden, T., Koleva, B., & Greenhalgh, C. (2002). A toolkit for user re-configuration of ubiquitous domestic environments. *Companion to Proceedings of UIST 2002*.
- Ashton, K. (2009). That 'internet of things' thing. *RFiD Journal*, 22, 97-114.
- Bakker, S., van den Hoven, E., Eggen, B., & Overbeeke, K. (2012, February). Exploring peripheral interaction design for primary school teachers. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (pp. 245-252). ACM.
- Bakker, S., Van Den Hoven, E., & Eggen, B. (2013, February). FireFlies: physical peripheral interaction design for the everyday routine of primary school teachers. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction* (pp. 57-64). ACM.
- Barchetti, U., Bucciero, A., Benedittis, T., Macchia, F., Mainetti, L., & Tamborino, A. (2009, July). MoWeT: A Configurable Framework to Support Ubiquitous Location-Aware Applications. In *Ubiquitous, Autonomic and Trusted Computing*, 2009. UIC-ATC'09. Symposia and Workshops on (pp. 75-82). IEEE.
- Benford, S., Crabtree, A., Flinham, M., Drozd, A., Anastasi, R., Paxton, M., ... & Row-Farr, J. (2006). Can you see me now?. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 13(1), 100-133.
- Benford, S., Flinham, M., Drozd, A., Anastasi, R., Rowland, D., Tandavanitj, N., ... & Sutton, J. (2004). Uncle Roy All Around You: Implicating the city in a location-based performance. *Proc. Advances in Computer Entertainment (ACE 2004)*, 21, 47.
- Benford, S., Magerkurth, C., & Ljungstrand, P. (2005). Bridging the physical and digital in pervasive gaming. *Communications of the ACM*, 48(3), 54-57.
- Bereiter, C., and Scardamalia, M. (1989). Intentional learning as a goal of instruction. In L. Resnick (Ed), *Knowing, Learning, and Instruction* (pp. 361-392). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

- Bielaczyc, K., & Collins, A. (1999). Learning communities in classrooms: A reconceptualization of educational practice. *Instructional-design theories and models: A new paradigm of instructional theory*, 2, 269-292.
- Blackstock, M., Kaviani, N., Lea, R., & Friday, A. (2010, November). MAGIC Broker 2: An open and extensible platform for the Internet of Things. In *Internet of Things (IOT)*, 2010 (pp. 1-8). IEEE.
- Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). *How people learn*. Washington, DC: National Academy Press.
- Branton, C., Ullmer, B., Wiggins, A., Rogge, L., Setty, N., Beck, S. D., & Reeser, A. (2013, June). Toward rapid and iterative development of tangible, collaborative, distributed user interfaces. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems* (pp. 239-248). ACM.
- Bronsted, J., Hansen, K. M., & Ingstrup, M. (2010). Service composition issues in pervasive computing. *Pervasive Computing, IEEE*, 9(1), 62-70.
- Brooks, F. P. (1975). *The mythical man-month* (Vol. 1995). Reading, MA: Addison-Wesley.
- Brown, A. L., & Campione, J. C. (1996). *Psychological theory and the design of innovative learning environments: On procedures, principles, and systems*. Lawrence Erlbaum Associates, Inc.
- Buschmann, F. (2010). Featuritis, Performitis, and Other Diseases. *IEEE software*, 27(1), 10-11.
- Cheverst, K., Davies, N., Mitchell, K., Friday, A., & Efstratiou, C. (2000, April). Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 17-24). ACM.
- Chi, M. T., Roscoe, R. D., Slotta, J. D., Roy, M., & Chase, C. C. (2012). Misconceived causal explanations for emergent processes. *Cognitive science*, 36(1), 1-61.
- Clancey, W. (1997). *Situated Cognition: On Human Knowledge and Computer Representations*. Cambridge, MA: Cambridge University Press.
- Clark, A. (1997). *Being There: Putting Brain Body and World Together Again*. Cambridge, MA: MIT Press.

- Cober, R., Fong, C., Gnoli, A., Silva, B. L., Lui, M., Madeira, C., ... & Tissenbaum, M. (2012). Embedded Phenomena for Knowledge Communities: Supporting complex practices and interactions within a community of inquiry in the elementary science classroom. In *Proceedings of the 10th International Conference of the Learning Sciences* (Vol. 2, pp. 64-71).
- Colella, V., and Borovoy, R. (1997, December). Adding a Thin Layer of Computation to Face-to-Face Collaboration. Presented at the *Computer Supported Collaborative Learning Conference (CSCL)*. Toronto, Ontario.
- Colella, V. (2000). Participatory simulations: Building collaborative understanding through immersive dynamic modeling. *The Journal of the Learning Sciences*, 9(4), 471-500.
- Curry, E. (2004). Message-oriented middleware. *Middleware for communications*, 1-28.
- Davis, A. M. (1995). Software prototyping. *Advances in computers*, 40, 39-63.
- Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012, February). Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (pp. 1277-1286). ACM.
- Dillenbourg, P., & Jermann, P. (2010). Technology for classroom orchestration. In *New science of learning* (pp. 525-552). Springer New York.
- Dourish, P. (2004). *Where the action is: the foundations of embodied interaction*. MIT press.
- Duschl, R. A., Schweingruber, H. A., & Shouse, A. W. (Eds.). (2007). *Taking Science to School: Learning and Teaching Science in Grades K-8*. National Academies Press.
- Enyedy, N., Danish, J. A., Delacruz, G., & Kumar, M. (2012). Learning physics through play in an augmented reality environment. *International Journal of Computer-Supported Collaborative Learning*, 7(3), 347-378.
- Erbad, A., Blackstock, M., Friday, A., Lea, R., & Al-Muhtadi, J. (2008, March). Magic broker: A middleware toolkit for interactive public displays. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on* (pp. 509-514). IEEE.
- Fischer, F., & Dillenbourg, P. (2006). Challenges of orchestrating computer-supported collaborative learning. Paper presented at the *87th Annual Meeting of the American Educational Research Association (AERA)*, San Francisco, California, USA.

- Garlan, D., Siewiorek, D. P., Smailagic, A., & Steenkiste, P. (2002). Project aura: Toward distraction-free pervasive computing. *Pervasive Computing*, IEEE, 1(2), 22-31.
- Glenberg, A. (1997). What memory is for. *Behavioral and Brain Sciences* 20, 1-55.
- Glenberg, A. (1999). Why Mental Models Must Be Embodied. In *Mental Models in Discourse Processing and Reasoning*, Rickheit, G. and Habel, C. (eds). New York: Elsevier.
- Gnoli, A. (2012) Connecting teachers with information in real-time, embodied, whole-class, collaborative inquiry. Presented at *10th International Conference on Learning Sciences* (Sidney, Australia, July 2 - 6, 2012). ICLS 2012.
- Gnoli, A., Perritano, A., Guerra, P., Lopez, B., Brown, J., & Moher, T. (2014). Back to the future: embodied classroom simulations of animal foraging. In *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction* (pp. 275-282). ACM.
- Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., & Wang, M. (2011). Proxemic interactions: the new ubicomp?. *interactions*, 18(1), 42-50.
- Greenhalgh, C. (2002, July). EQUIP: An extensible platform for distributed collaboration. In *The Second Workshop on Advanced Collaborative Environments*.
- Greenhalgh, C., Izadi, S., Mathrick, J., Humble, J., & Taylor, I. (2004). A Toolkit to Support Rapid Construction of Ubicomp Environments. *Proceedings of UbiSys*.
- Gresalfi, M.S., Barab, S., Siyahhan, S., & Christensen, T. (2009). Virtual worlds, conceptual understanding, and me: Designing for Critical engagement. *On the Horizon* 17(1), 21-34.
- Grimm, R. (2004). One. world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, (3), 22-30.
- Guo, B., Fujimura, R., Zhang, D., & Imai, M. (2012). Design-in-play: improving the variability of indoor pervasive games. *Multimedia Tools and Applications*, 59(1), 259-277.
- Gutierrez, L., Nikolaidis, I., Stroulia, E., Gouglas, S., Rockwell, G., Boechler, P., ... & King, S. (2011, March). far-play: A framework to develop augmented/alternate reality games. In *Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2011 IEEE International Conference on (pp. 531-536). IEEE.

- Gutierrez, L., Stroulia, E., & Nikolaidis, I. (2012). fAARS: a platform for location-aware trans-reality games. In *Entertainment Computing-ICEC 2012* (pp. 185-192). Springer Berlin Heidelberg.
- Gutiérrez, L. A. G. (2012). *The FAARS Platform: For Augmented Alternate Reality Services and Games* (Master thesis, University of Alberta).
- Han, S. W., Yoon, Y. B., Youn, H. Y., & Cho, W. D. (2004, May). A new middleware architecture for ubiquitous computing environment. In *Software Technologies for Future Embedded and Ubiquitous Systems, 2004. Proceedings. Second IEEE Workshop on* (pp. 117-121). IEEE.
- Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., & Jansen, E. (2005). The gator tech smart house: A programmable pervasive space. *Computer*, 38(3), 50-60.
- Hewitt, C., Bishop, P., & Steiger, R. (1973, August). A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence* (pp. 235-245). Morgan Kaufmann Publishers Inc.
- Humble, J., Crabtree, A., Hemmings, T., Åkesson, K. P., Koleva, B., Rodden, T., & Hansson, P. (2003, January). "Playing with the Bits" User-configuration of Ubiquitous Domestic Environments. In *UbiComp 2003: Ubiquitous Computing* (pp. 256-263). Springer Berlin Heidelberg.
- Hutchins, E. (1995). *Cognition in the Wild*. MIT press.
- Jackson, M., Crouch, S., & Baxter, R. (2013). *Software Evaluation Guide*. Software Sustainability Institute.
- Jarvis, T., & Pell, A. (2005). Factors influencing elementary school children's attitudes toward science before, during, and after a visit to the UK National Space Centre. *Journal of research in science teaching*, 42(1), 53-83.
- Johnson, M. L. (1987). *The body in the mind: The bodily basis of meaning, imagination, and reason*. Chicago University Press.
- Kollar, I., Fischer, F., and Slotta, J. D. (2007). Internal and external scripts in computer-supported collaborative inquiry learning. *Learning & Instruction*, 17(6), 708-721.
- Krasner, G. E., & Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3), 26-49.

- Krajcik, J. S., & Berg, C. (1987). Exemplary software for the science classroom. *School Science and Mathematics*, 87(6), 494-500.
- Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, 36(6), 47-56.
- Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge university press.
- Lee, C., Nordstedt, D., & Helal, S. (2003). Enabling smart spaces with OSGi. *Pervasive Computing, IEEE*, 2(3), 89-94.
- Lindgren, R., & Johnson-Glenberg, M. (2013). Emboldened by Embodiment Six Precepts for Research on Embodied Learning and Mixed Reality. *Educational Researcher*, 42(8), 445-452.
- Lindt, I., Ohlenburg, J., Pankoke-Babatz, U., & Ghellal, S. (2007). A report on the crossmedia game epidemic menace. *Computers in Entertainment (CIE)*, 5(1), 8.
- Lui, M., & Slotta, J. D. (2013). Exploring Evolutionary Concepts with Immersive Simulations. In *Proceedings of the 10th International Conference on Computer Supported Collaborative Learning* (pp. 304-311). ISLS.
- Lui, M., & Slotta, J. D. (2014). Immersive simulations for smart classrooms: exploring evolutionary concepts in secondary science. *Technology, Pedagogy and Education*, 23(1), 57-80.
- Luyten, K., & Coninx, K. (2005). Distributed user interface elements to support smart interaction spaces. In *Multimedia, Seventh IEEE International Symposium on* (pp. 8-pp). IEEE.
- Machado, C., Silva, E., Batista, T., Leite, J., & Yumi Nakagawa, E. (2013, October). Architectural elements of ubiquitous systems: A systematic review. In *ICSEA 2013, The Eighth International Conference on Software Engineering Advances* (pp. 208-213).
- Marquardt, N., Diaz-Marino, R., Boring, S., & Greenberg, S. (2011, October). The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 315-326). ACM.
- Moher, T., Hussain, S., Halter, T., & Kilb, D. (2005). RoomQuake: embedding dynamic phenomena within the physical space of an elementary school classroom. In

- CHI'05 Extended Abstracts on Human Factors in Computing Systems* (pp. 1665-1668). ACM.
- Moher, T. (2006). Embedded phenomena: supporting science learning with classroom-sized distributed simulations. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 691-700). ACM.
- Moher, T. (2008, June). Learning and participation in a persistent whole-classroom seismology simulation. In *Proceedings of the 8th international conference on International conference for the learning sciences-Volume 2* (pp. 82-90). International Society of the Learning Sciences.
- Moher, T., Uphoff, B., Bhatt, D., López Silva, B., & Malcolm, P. (2008). WallCology: Designing interaction affordances for learner engagement in authentic science inquiry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 163-172). ACM.
- Moher, T., Wiley, J., Jaeger, A., Silva, B. L., Novellis, F., & Kilb, D. (2010, June). Spatial and temporal embedding for science inquiry: an empirical study of student learning. In *Proceedings of the 9th International Conference of the Learning Sciences-Volume 1* (pp. 826-833). International Society of the Learning Sciences.
- Moher, T., Novellis, F., Lopez Silva, B., Gnoli, A. (2012). Embodied science practices in hybrid spaces. In O. Smortal (Chair), *Hybrid spaces for science learning: New demands and opportunities for research*. In *Proceedings of the 10th International Conference of the Learning Sciences*. ISLS.
- Novellis, F., & Moher, T. (2011). How real is 'real enough?': designing artifacts and procedures for embodied simulations of science practices. In *Proceedings of the 10th International Conference on Interaction Design and Children* (pp. 90-98). ACM.
- O'Brien, T., Moser, M., Casey, J., Fox, B., Zyl, J., Redmond, E., & Shatzer, L. (2010). *Maven: The Complete Reference*. Online Book.
- Penuel, W. R., Roschelle, J., Crawford, V., Shechtman, N., & Abrahamson, L. (2004). Advancing Research on the Transformative Potential of Interactive Pedagogies and Classroom Network. In *Workshop Report, SRI International and Better Education Foundation*.
- Peppler, K., Danish, J., Zaitlen, B., Glosson, D., Jacobs, A., & Phelps, D. (2010, June). BeeSim: leveraging wearable computers in participatory simulations with young

- children. In *Proceedings of the 9th International Conference on Interaction Design and Children* (pp. 246-249). ACM.
- Perez de Almeida, R. A., Blackstock, M., Lea, R., Calderon, R., do Prado, A. F., & Guardia, H. C. (2013, September). Thing broker: A twitter for things. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication* (pp. 1545-1554). ACM.
- Price, S., Rogers, Y., Scaife, M., Stanton, D., & Neale, H. (2003). Using 'tangibles' to promote novel forms of playful learning. *Interacting with computers*, 15(2), 169-185.
- Raatikainen, M., Komssi, M., Dal Bianco, V., Kindstom, K., & Jarvinen, J. (2013, July). Industrial Experiences of Organizing a Hackathon to Assess a Device-centric Cloud Ecosystem. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual* (pp. 790-799). IEEE.
- Raychoudhury, V., Cao, J., Kumar, M., & Zhang, D. (2013). Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing*, 9(2), 177-200.
- Rodden, T., Crabtree, A., Hemmings, T., Koleva, B., Humble, J., Åkesson, K. P., & Hansson, P. (2004, August). Between the dazzle of a new building and its eventual corpse: assembling the ubiquitous home. In *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques* (pp. 71-80). ACM.
- Rodríguez-Domínguez, C., Benghazi, K., Noguera, M., Garrido, J. L., Rodríguez, M. L., & Ruiz-López, T. (2012). A communication model to integrate the request-response and the publish-subscribe paradigms into ubiquitous systems. *Sensors*, 12(6), 7648-7668.
- Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002). Gaia: a middleware platform for active spaces. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4), 65-67.
- Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002). A middleware infrastructure for active spaces. *IEEE pervasive computing*, 1(4), 74-83.
- Roschelle, J., & Pea, R. (2002). A walk on the WILD side: How wireless handhelds may change computer-supported collaborative learning. *International Journal of Cognition and Technology*, 1(1), 145-168.

- Rummel, N., and Spada, H. (2005). Learning to collaborate: An instructional approach to promoting collaborative problem-solving in computer-mediated settings. *Journal of the Learning Sciences*, 14(2), 201-241.
- Salber, D., Dey, A. K., & Abowd, G. D. (1999, May). The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 434-441). ACM.
- Scardamalia, M., & Bereiter, C. (1994). Computer support for knowledge-building communities. *The journal of the learning sciences*, 3(3), 265-283.
- Scardamalia, M., and Bereiter, C. (2003). Knowledge building environments: Extending the limits of the possible in education and knowledge work. In A. DiStefano, K.E. Rudestam, & R. Silverman (Eds.), *Encyclopedia of distributed learning*. Thousand Oaks, CA: Sage Publications.
- Schreiner, M., Rädle, R., Jetter, H.-C., & Reiterer, H. (2015). Connichiwa: A Framework for Cross-Device Web Applications. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 2163–2168). New York, NY, USA: ACM. doi:10.1145/2702613.2732909
- Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft Press.
- Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25(4), 557-572.
- Sibley, B. A., & Etnier, J. L. (2003). The relationship between physical activity and cognition in children: a meta-analysis. *Pediatric Exercise Science*, 15(3), 243-256.
- Sintoris, C., Yiannoutsou, N., Ortega-Arranz, A., López-Romero, R., Masoura, M., Avouris, N., & Dimitriadis, Y. (2014, November). TaggingCreaditor: A tool to create and share content for location-based games for learning. In *MLCIOS: Special Session "Mobile Learning in Cultural Institutions and Open Spaces"*, IMCL2014.
- Slotta, J. D., & Aleahmad, T. (2009). WISE technology lessons: Moving from a local proprietary system to a global open source framework. *Research and Practice in Technology Enhanced Learning*, 4(02), 169-189.
- Slotta, J. D., Aleahmad, T., & van Joolingen, W. (2009, June). Toward a technology community in the learning sciences. In *Proceedings of the 8th International Conference on Computer Supported Collaborative Learning, Vol.2* (pp. 12-14). ISLS.

- Slotta J D and Najafi H (2010), Knowledge Communities in the Classroom. In Penelope Peterson, Eva Baker, Barry McGaw, (Eds), *International Encyclopedia of Education*. Volume 8, pp. 189-196. Oxford: Elsevier.
- Sørensen, H., & Kjeldskov, J. (2013, December). Moving Beyond Weak Identifiers for Proxemic Interaction. In *Proceedings of International Conference on Advances in Mobile Computing & Multimedia* (p. 18). ACM.
- Streitz, N. A., Geißler, J., Holmer, T., Konomi, S. I., Müller-Tomfelde, C., Reischl, W., ... & Steinmetz, R. (1999, May). i-LAND: an interactive landscape for creativity and innovation. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 120-127). ACM.
- Streitz, N. A., Tandler, P., Müller-Tomfelde, C., & Konomi, S. I. (2001). Roomware: Towards the Next Generation of Human-Computer: Interaction based on an Integrated Design of Real and Virtual Worlds. *Human-Computer Interaction in the New Millenium*, Addison Wesley, 551-576.
- Strong, W. B., Malina, R. M., Blimkie, J. R., Daniels, S., Dishman, R., Gutin, B., ... & Trudeau, F. (2005). Physical activity recommendations for school-age youth. *J Pediatr*, 146, 732-737.
- Tandler, P. (2001, January). Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices. In *Ubicomp 2001: Ubiquitous Computing* (pp. 96-115). Springer Berlin Heidelberg.
- Thompson, M., & Moher, T. (2006). HelioRoom: Problem-solving in a whole class visual simulation. In *Proceedings of the 7th international conference on Learning sciences* (pp. 1000-1001). International Society of the Learning Sciences.
- Tissenbaum, M., & Slotta, J. D. (2015). Scripting and Orchestration of Learning Across Contexts: A Role for Intelligent Agents and Data Mining. In *Seamless Learning in the Age of Mobile Connectivity* (pp. 223-257). Springer Singapore.
- Varshavsky, A., & Patel, S. (2009). Location in ubiquitous computing. *Ubiquitous computing fundamentals*, 285-320.
- Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard university press.
- Weinberger, A., Ertl, B., Fischer, F., and Mandl, H. (2005). Epistemic and social scripts in computer- supported collaborative learning. *Instructional Science*, 33, 1–30.

- Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 265(3), 94-104. DOI:10.1038/scientificamerican0991-94
- Winn, W.D. (2003). Learning in artificial environments: Embodiment, embeddedness and dynamic adaptation. *Technology, Instruction, Cognition and Learning* 1, 87-114.
- Wohlin, C., Höst, M., & Henningsson, K. (2003). Empirical research methods in software engineering. In *Empirical methods and studies in software engineering* (pp. 7-23). Springer Berlin Heidelberg.
- Yang, H., Jansen, E., Helal, A., & Mann, W. (2006). An IDE for programmable pervasive spaces based on a context-driven programming model. *4th IEEE Intl. Conference on Pervasive Computing and Communications*.
- Yau, S. S., Karim, F., Wang, Y., Wang, B., & Gupta, S. K. (2002). Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3), 33-40.

VITA

- Name: Alessandro Gnoli
- Education: B.S., Computer Science, Politecnico di Milano, Milan, Italy, 2006
M.S., Computer Science, Politecnico di Milano, Milan, Italy, 2008
M.S., Computer Science, University of Illinois at Chicago, Illinois, 2009
Ph.D, Computer Science, University of Illinois at Chicago, Illinois, 2015
- Professional membership: American Education Research Association
Association for Computer Machinery
Institute of Electrical and Electronics Engineers
International Society of the Learning Sciences
- Professional experience: Graduate Research Assistant
Computer Science Department, Learning Technologies Group
January 2008 - May 2015
- Graduate Research Assistant
Learning Sciences Research Institute, Digital Literacy project
January 2010 - May 2010
- Teaching Assistant, CS 440 Software Engineering
Spring semester 2010
- Teaching Assistant, CS 335 Computer Ethics
Spring semester 2009
- Publications: Moher, T. Slotta, J. Acosta, A. Gnoli, A. ... Perritano, A. (2015)
Knowledge Construction in the Instrumented Classroom: Supporting Student Investigations of Their Physical Learning Environment. In *Proceedings of the 11th International Conference on Computer Supported Collaborative Learning* (pp. 631-638). ISLS
- Gnoli, A. Perritano, A. Guerra, P. Lopez, B. Brown, J. Moher, T. (2014)
Back to the Future: Embodied Classroom Simulations of Animal

Foraging. In *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction* (pp. 275-283). ACM.

Gnoli, A. Moher, T. (2013) Providing Teachers With Real-Time Feedback on the Fidelity of Science Practice. Paper presented at the *Annual Conference of the American Educational Research Association* (San Francisco, CA, April 27 – May 1, 2013). AERA 2013.

Smørdal, O. Slotta, J. Krange, I. Moher, T. Novellis, F. Gnoli, A. Silva, B. L. Lui, M. Jornet, A. Jahreie, C. F. (2012) Hybrid spaces for science education. In *Proceedings of the 10th International Conference on Learning Sciences* (Sidney, Australia, July 2 - 6, 2012). ICLS 2012.

Cober, R. Fong, C. Gnoli, A. Silva, B. L. Lui, M. Madeira, C. McCann, C. Moher, T. Slotta, J. Tissenbaum, M. (2012) Embedded Phenomena for Knowledge Communities: Supporting complex practices and interactions within a community of inquiry in the elementary science classroom. In *Proceedings of the 10th International Conference on Learning Sciences* (Sidney, Australia, July 2 - 6, 2012). ICLS 2012.

Gnoli, A. (2012) Connecting teachers with information in real-time, embodied, whole-class, collaborative inquiry. Accepted at Doctoral Consortium of the *10th International Conference on Learning Sciences* (Sidney, Australia, July 2 - 6, 2012). ICLS 2012.

Madeira, C.A. Gnoli, A., Messina, R. (2012) Fostering an adaptive nature for Teacher Practice: Technology Supports in a Co-Design Community. Paper presented at the *Annual Conference of the American Educational Research Association* (Vancouver, BC, April 13 - 17, 2011). AERA 2012.

Slotta J., Tissenbaum. M, Lui M., Alagha I, Burd E., Higgins S., Mercier E., Fischer F., Pilz F., Kollar I., Moher T., Gnoli A., Jaeger A., Wiley J., López Silva B., Evans M., Motto A., Brunger A., Crider J., Wilkins J. (2011) Embedding CSCL in Classrooms: Conceptual and Methodological Challenges of Research on New Learning Spaces. In *Proceedings of the 9th International Conference on Computer-Supported Collaborative Learning* (Hong Kong, China, July 4 - 9, 2011). CSCL 2011.

Moher, T., Slotta, J., & Gnoli, A. (2011). Embedded phenomena: Rethinking technology support for complex collaborative activity

structures in classrooms. Paper presented at the *Annual Conference of the American Educational Research Association* (New Orleans, LA, April 8 - 12, 2011). AERA 2011.

Jaeger, A., Moher, T., Wiley, J., Malcolm, P., López Silva, B., Gnoli, A., and Brown, J. (2009). WallCology: Using Embedded Phenomena to Motivate Learning About Dynamic Ecosystems. Paper presented at the *Annual Conference of the American Educational Research Association* (San Diego, CA, April 13 - 17, 2009). AERA 2009