A Reactive Behavioral System for the Intelligent Room

by

Ajay A. Kulkarni

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Author ....
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by... ....
Dr. Howard Shrobe
Principal Research Scientist
Thesis Supervisor

Certified by... ....
Prof. Brian Williams
Associate Professor
Thesis Supervisor

Accepted by....... ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Reactive Behavioral System for the Intelligent Room

by

Ajay A. Kulkarni

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Traditional computing interfaces have drawn users into a world of windows, icons and pointers. Pervasive computing believes that human-computer interaction (HCI) should be more natural: computers should be brought into our world of human discourse. The Intelligent Room project shares this vision. We are building an Intelligent Environment (IE) to allow for more natural forms of HCI. We believe that to move in this direction, we need our IE to respond to more than just direct commands; it also needs to respond to implicit user commands, such as body language, behavior, and context, just as another human would. This thesis presents *ReBa*, a context-aware system to provide the IE with this type of complex behavior, in reaction to user activity.

Thesis Supervisor: Dr. Howard Shrobe
Title: Principal Research Scientist

Thesis Supervisor: Prof. Brian Williams
Title: Associate Professor

Acknowledgments

I'd like to thank my thesis advisors, Howie Shrobe and Brian Williams, for providing me with guidance and inspiration during this whole process. I would also like to express gratitude to my academic advisor, Jim Glass, who has helped me navigate through the MIT curriculum for the past five years.

I'd like especially to thank Krzysztof Gajos, who has read multiple drafts of all my papers (and thesis), helping me articulate my ideas. He has been an outstanding mentor and a good friend.

In addition, Kimberle Koile and Bob Laddaga have also read drafts of this thesis, providing valuable editing and advice.

I'd also like to thank Mike Freedman and Nick Feamster, two great friends and highly motivating colleagues. Our apartment was often the home of memorable debates about everything within MIT and without, from United States politics, to the world economy, to the Middle East (and yes, even computer science).

Finally, I'd like to thank my parents and my sister Amita. They have always provided me with support, encouragement, and laughter. They are the best parents and sister that I could have ever hoped for.

Contents

List of Figures

Actioni contrariam semper et aeqalem esse reactionem

- Isaac Newton, Principia

What flows like water, reflects like a mirror, and responds like an echo?

- Benjamin Hoff, The Tao of Pooh

Chapter 1

Reactive Behavior

1.1 Introduction

For about the past twenty years, we have incorporated computers into our lives by placing them on our desks and laps; to use these desktop and laptop computers, we are forced to enter their world of windows, icons and pointers. At the Intelligent Room, an experimental *Intelligent Environment* (IE) we have built, we believe that human-computer interaction should be the other way around: the computer should be brought into our world, into the human world of discourse and behavior. We should be allowed to use speech, gesture and movement to interact with our computing environment.

To support this type of interaction, an IE needs to respond to its users. Most of the previous research in this area has centered on making the IE respond to explicit user commands. We can improve on this work by making it respond to more implicit commands; the IE needs to be capable of reacting to user behavior. To perform this function, the IE needs to have an understanding of context and be able to situate user actions within their context. In this thesis, I present a *reactive behavioral system* called ReBa, which enables the Intelligent Room to exhibit this complex behavior in reaction to user activity. ReBa has two components: a software infrastructure to simplify the creation of room reactions, and a set of room reactions that use this infrastructure. ReBa is already implemented and incorporated into the Intelligent Room.

1.2 An Illustration

Anthropocentric computing requires the thing doing the computation, the technologically enhanced environment, to react to human user actions. The following scenario illustrates how simple environment reactions can aid the user in his or her tasks:

Alice walks into an empty conference room to set up for her evening presentation. The room, sensing that a new user has entered, turns on the lights. She connects the video output of her laptop to the room's display system. As soon as she displays her slides on her laptop, the room recognizes that she is about to give a presentation, and also displays her slides on the main wall. She is ready to start the presentation.

This scenario makes the design and implementation of a room reaction seem like a three-step process: *wait* for a user action, *sense* a user action, and *respond* to that user action by changing the state of the room. Indeed, this process has been the basis of most interactive computer use so far, except that normally human actions were limited to typing and mouse clicks. An earlier attempt at Intelligent Room reactivity also used this method, extending it to handle actions like gestures and movement. Section 2.1 describes this system.

1.2.1 The Problem

If we introduce a few changes into our scenario, *wait-sense-respond* breaks down, creating a few undesirable effects. Imagine the following alterations to our scenario:

- *Daylight* - If Alice enters the room in the morning, natural lighting is most likely a better alternative than artificial. At the very least, the room should open the drapes instead of turning on the lights.

- *Non-Empty Room* - If Alice enters the room in the middle of another presentation, turning on the lights would disrupt the individuals already present. The room should recognize that not affecting the lights is more desirable than turning them on.

- *Multiple Displays* - If the conference room has multiple displays–say, one at the end of the room for formal meetings, another on a side wall for more collaborative meetings–how does the room know which display to use? If the room knew whether Alice was participating in a formal meeting or in a more casual one, it could decide between the two displays.

- *Different Users* - If users vary in their expectations of the room, then different individuals using the room should trigger different room reactions. Perhaps Alice likes artificial lighting and using the main wall. Bob, on the other hand, may prefer natural lighting when possible and using the side wall.

- *Multiple Users* - If there are multiple users in the room, each with different preferences, how does the room decide which reactions to trigger? The room needs a way to resolve conflicts.

What these examples show is that designing an effective reactive system for an intelligent space is difficult. One could try to encode these exceptions into a *wait-sense-respond* system, but it would be impossible to anticipate all exceptions. Also, including exceptions in such a system misses the actual problem: the system does not understand the context of user actions. If the room could situate a user's actions in a context, e.g., time of day, current room usage, type of meeting, and user identity, and react to the user according to that context, then the above alterations to our scenario could be handled easily. Thus, our reactive system needs to be context-aware.

1.2.2 What this is, What this is Not

My dictionary defines "behavior" as "anything that an organism does involving action and response to stimulation." We want the Intelligent Room to exhibit actions in response to user stimuli. These stimuli, however, are more than just commands; we want a proactive room that reacts to user actions and intentions, commands and body language. This subsection elaborates on what ReBa should be responsible for, and what it should not address.

Our system will provide reactivity to user behavior and context. We believe that context-based reactions are an important part of interpersonal communication. Humans often interact with each other through implicit signals, which then may lead to more explicit discourse. For example, to get someone's attention, you may make eye contact, raise your hand, or move closer. On the other hand, an explicit statement like, "May I speak with you," is unnatural, and usually reserved for more formal instances. We find this conduct natural and expect it from others; we should also expect this sort of behavior from our Intelligent Room.

Our reactive system will not perform three functions that it needs; it will rely on other systems to collect perceptual data, manage resources, and control access to resources.

Collection of Perceptual Data - Our system will not collect perceptual data about users and the state of the room. We believe that there should be a separation between the acquisition of this information and its use. As a result, we rely on other systems for head, arm, and person tracking [29, 10], for location information [26], for natural language processing [24], for device state detection, and for pressure and temperature sensing. From these perceptual systems, our reactive systems will receive high level inferences in the form of events. There is a basic ontology to these events. The abstraction of the perceiving systems from the reacting ones is discussed further in section 2.3.

Management of Resources - Our system will not use information about its state for the purpose of allocating resources. Resource management is in itself a complicated problem of service discovery, delegation, and arbitration [22, 21]. When our reactive system uses a resource to react or to provide a service to the user, it will rely on a separate resource manager [19].

Access Control Mechanism - Our system will not act as an access control mechanism (ACM). ACMs restrict resources from unauthorized users. An effective ACM in an IE needs to include contextual information, such as the location of the user and the current mode of the environment, into its rules. More information can be found in [41].

1.3 Intelligent Environments

The notion of Intelligent Room reactivity is inherent within the design of an Intelligent Environment, within the dream of *pervasive computing*, and even within the dream of human-computer interaction (HCI) in general. An IE is an interactive space, technologically enhanced to help people with tasks that historically have not involved–or have unsuccessfully involved–computer assistance. IEs are designed to allow users to interact with their environment as they would with another human: using speech, gesture, movement, and context [7].

The late Mark Weiser is often credited as the father of the vision of embedding technology in everyday objects, which inspired the field of *ubiquitous computing.* In his "Computing for the 21st Century," Weiser illustrates that the best technologies "are those that disappear," those that "weave themselves into the fabric of everyday life until they are indistinguishable from it" [43]. For computers to be truly helpful, he argues, they need to be embedded in our environment.

MIT Project Oxygen extends this vision one step further: it wants to make technology as pervasive "as the air we breathe" [11]. Instead of only embedding technology in our surroundings, it argues, computation should be integrated into our lives as unimposing elements of assistance.

1.3.1 The Intelligent Room

At the Intelligent Room Project, we share this vision of pervasive computing. Enhanced with cameras, microphones, sensors, a variety of devices, and several computers, the Intelligent Room is an experiment in multi-modal human-computer interaction.

The infrastructure in the Intelligent Room is implemented in the multi-agent system *Metaglue* [33, 42, 9]. An extension of the *Java* programming language, Metaglue supports autonomous agents with a fault tolerant communication infrastructure, agent naming-space organization, and a persistent storage mechanism.

In the Intelligent Room, a user currently has the following choices for interaction: speech (via the Laureate and ViaVoice recognition systems), gesture (via the Mimio^{TM}pen mouse), movement (via vision tracking systems), and device manipulation (via controllers written in software). We are working on incorporating several other components: context-based speech using the *Galaxy System* [14]; a framework for maintaining security and protecting privacy [41]; an expressive and emotive user interface called *SAM* [30]; and the topic of this thesis, a context-aware reactive system. A description of several of these components can be found in [23].

Metaglue has been used in various physical spaces, resulting in a variety of Intelligent Room implementations: a conference room, an office, a living room, a mobile hand-held device, and even an anonymous information kiosk. Our research involves interaction in and across each of these types of physical spaces.

Current Intelligent Room applications that are supported by the Metaglue infrastructure include:

- *Meeting Manager* - Provides assistance in running and recording meetings [32, 31]

- *FIRE* - Multi-modal information retrieving and browsing system [20]
- *Assist* - Sketch interpretation tool for designing mechanical systems [1]
- *News Wall* - Presentation and organization of various news categories
- *Map System* - Multimodal tool for geographic information
- *Interactive Boggle®* - Multi-space implementation of the popular Parker Brothers® board game.

1.4 Understanding Intelligence

If we try to find models of the type of flexible reactivity that our scenarios illustrate, we are led back to ourselves. There is a strong resemblance between the way we imagine our IEs responding and the way a person does; we look to theories of human behavior and robots for clues about building reactive IEs.

1.4.1 Human Behavior

Human behavior remains our best model of flexible reactive behavior. Even the simplest of human actions requires complex systems. Imagine yourself walking across a room. (Even better, walk across the room right now. I'll wait for you to return.) Your lower body, a collection of about 60 bones, 50 muscles, and various tendons, helps you move from one point in the room to another. In addition, you need to remain upright. Carrying your body vertically is an incredible test of balance, involving your spine and even the bones in your inner ear. Finally, you rely on your visual and tactile perceptions to manage obstacles or even slight changes in the level of the floor. Human perception apparently relies on the integration of modalities and *cross-modal influence*: the sharing of sensory information across modalities at various levels of perception [8]. The ability for all these components to work together is what allows you to move.

To explain these complex human systems, Marvin Minsky proposes his theory of "The Society of Mind" [28]. He argues that the human mind is composed of smaller, simpler processes, called "agents," where each agent is designed to carry out a task that requires no thought. Human intelligence and behavior emerge from the interaction of these non-intelligent components. We have adopted this viewpoint in our work.

1.4.2 Mobile Robots

The majority of research in reactive systems has been centered around creating a *robot*, often described as a mechanical device, sometimes resembling a human, that is capable of performing a veriety of simple and complex human tasks. The word originates from the Czech "robota," (forced or repetitive labor, hard work), and the English translation of Karel Capek's 1921 play, "R.U.R" (Rossum's Universal Robots). The term *robotics*, denoting the scientific field for the study of the creation of robots, was first used by Isaac Asimov, a prolific writer of science fiction and natural science, not to mention fantasy, mystery, and humor.

While robots and other forms of artificial life have long existed in fiction, computing advances of the second half of the 20th century have started to allow the research community to approach this fiction. Over those years, there has been a variety of research in *mobile robots*, robots with the ability to move through and/or explore their surroundings.

For a mobile robot to move from one point to another or to wander around its surroundings, it needs to navigate through its environment, recognizing and avoiding obstacles. Earlier approaches used a top-down approach, first *sensing* the world, *modeling* it, *planning* a path, and finally *acting*. Later approaches started to use a bottom-up approach, focusing on reacting to the environment rather than to a model of the environment. Rodney Brooks embraced this vision with his *subsumption* architecture [3], which relies on creating complex behavior through the interactions of simpler layers of control mechanisms. According to Brooks, instead of trying to model the world, mobile robots should just respond to it.

A more recent approach, called the *RAP* system [16, 15], incorporates the compositional aspects of subsumption with the planning of the earlier systems. It uses modular reactive action packages (RAPs) as building blocks to construct a plan for reaction. The system carries out the actions in different situations, despite most problems that may arise.

These lessons are useful to us in our design of a reactive system for the Intelligent Room. Let's imagine a robot designed to help humans in their day-to-day tasks, except that our robot can't be seen. When a user wants to turn on the lights, the robot turns it on for her. When she wants to display her slides onto a wall, the robot takes the presentation bits and routes them to one of the projectors for her. When she wants to leave the room, the robot turns off the projector and the lights for her.

Our robot however is confined to the room: it is immobile. In effect, the Intelligent Room is an "immobile robot" [44]; some of the lessons learned in the study of mobile robots can be applied to research in room reactivity. Indeed, the roots of the current Intelligent Room project lie in "Hal," an earlier IRoom implementation which was loosely modeled after the humanized immobile robot "HAL 9000," depicted in Stanley Kubrick's "2001: A Space Odyssey."

Much of the literature on mobile robots is colored by the issues we do not face: locomotion, limits to the miniaturization of parts, minimizing computation and power consumption. Nevertheless, the similarities in the requirements of reactivity warrant further attention. This discussion is continued in section 2.2.

1.5 The Solution

In this thesis I describe a context-aware reactive system for the Intelligent Room. We have built this system using two guiding insights:

1. The Intelligent Room is an *immobile robot*

2. Intelligent behavior *emerges* from the composition of non-intelligent components

In the following chapters, I present *ReBa*, a reactive behavioral system for intelligent physical spaces. ReBa is primarily crafted out of five design principles:

- *Context-Awareness* - A reactive behavioral system needs to be aware of the context of user actions.

- *Conflict-Resolution* - A reactive behavioral system needs to arbitrate between conflicting reactions.

- *Adaptability* - A reactive behavioral system needs to be adaptable to the variety of intelligent physical spaces.

- *User-Centricity* - A reactive behavioral system needs to take into account the comfort and convenience of its users.

- *Evolvability* - A reactive behavioral system needs to be easily extended and improved.

In the next chapter, I analyze the limitations of a previous attempt at a reactive system in the Intelligent Room, looking towards existing work in mobile and immobile robots for assistance in understanding what qualities a reactive system should have. Following that chapter, I look back on the five characteristics of context-awareness, conflict-resolution, adaptability, user-centricity, and evolvability, and introduce a set of design decisions made for ReBa. Then, I present the ReBa system and compare it to other related research. Finally, I conclude with the contributions of this thesis.

The morning daylight appears plainer
When you put out your candle.
– Benjamin Franklin

Chapter 2

The Nature of the Problem

My last chapter glanced at the fields of Intelligent Environments, Robotics, and Human Intelligence, presenting five basic design characteristics for our reactive behavioral system. In this chapter, I look more closely at the nature of reactivity, first with a critique of an earlier attempt at an Intelligent Room reactive system, in which I identify a few qualities of a better system. Then I continue with a discussion of reactive systems in other research areas, namely Mobile Robots and Space Systems, in which I have found the inspiration for articulating what qualities our reactive system requires. I conclude with an examination of context and context-aware systems, describing how our reactive system should use this information. In this manner, I sow the seeds for more specific design requirements, which I present in the next chapter.

2.1 Lessons from the Past

The previous attempt at a reactive system in the Intelligent Room embodied a method of encoding room reactivity that was too simple. Through an analysis of its shortcomings we can understand what qualities a better system should possess. In this system, basic reactions were encoded in *Mess*, the Metaglue extension of *Jess* [18], a rule-based programming language written in Java. Rule-based programming languages are based on a database of *facts* and a set of *rules.* Rules are triggered on the presence (or absence) of certain facts. In the Intelligent Room, various conditions about the environment and users were encoded as facts. Similarly, a room reaction to those conditions was encoded as a rule. These facts and rules were organized into various files.

2.1.1 Example

The following scenario is an example of how this system would respond to user actions:

The intelligent conference room is currently unoccupied. Bob opens the door to the room, enters, and sits down to prepare for his 10 am presentation. The room, using the data from its break-beam on the door, its pressure mat in front of the door, and the camera pointed at the door, detects user entry and turns on the lights. Bob, however, prefers natural lighting to artificial, and since the sun is shining, gets up, turns off the lights, and asks the room to open the drapes.

He then connects his laptop to the room's display system, and loads his power-point slides. His laptop, which like the room is running Metaglue, detects the power-point slides and displays them on the main wall of the room. Bob, however, is just leading a discussion, and would rather display his slides on the side wall. He tells the room to switch the display to the side projector.

At 10 o'clock, Alice and Carol enter the room for the meeting. The room, detecting entry, turns on all the lights. Bob begins to tell the room to turn off the lights, but then stops, remembering that when he starts his presentation, the room will turn them off anyway. Bob then tells the room, "Start the Presentation," and the room responds by dimming the lights, to help Alice and Carol read Bob's slides. But Bob has already opened the drapes; the slides are still difficult to read. Bob asks the room to close the drapes, and then starts discussing his slides.

A few minutes later, David stumbles into the room and joins Alice and Carol at the table. The room, detecting entry, again turns on all the lights. David apologizes, gets up from the table, walks over to the light switch, and turns them off. Bob continues with his presentation.

In this example, room reactivity takes three forms: turning on the lights to user entry, displaying slides on the main wall, and dimming lights when a user starts a presentation. Each of these reactions would have been encoded in a different set of Mess rules.

2.1.2 Analysis

The room in this example is like an overzealous assistant; frankly, it is annoying. It turns on lights when it shouldn't, it displays your slides where it shouldn't, and it doesn't know when to dim the lights and when to close the drapes. It displays a simple, rigid understanding of user behavior, where an action is to be treated the same, regardless of what is occurring in the room. A better system would use the information about who you are, where you are, and what you are doing to adapt its reactions; our reactive system needs to be *context-aware.*

This reactive system is also not organized: rules are all over the place, and the system is a mess. To add a reaction to the room, a developer either modifies an existing rule or writes his own. Conscientious developers may take the time to check whether the new reaction belongs in an existing or in a new rule; even a few careless mistakes, like incorrectly naming a new rule, would not adversely affect the reaction. But if another developer wants to modify or even understand an existing reaction, trying to find the corresponding rules through poorly written code becomes a frustrating task. A better system would be *modular,* forcing developers to organize their reactions along specific guidelines.

This system cannot resolve conflicts between reactions; this problem is a major shortcoming of any reactive system. As more reactions are added, the probability of multiple reactions responding to the same user action in different ways increases. One rule may tell the room to turn on the lights when a user enters, a second to open the drapes, and a third to not affect the drapes or lights. Our reactive system needs to arbitrate between these conflicting reactions; it needs to have a mechanism for *conflict-resolution.*

A fourth problem with this system lies in the lack of its adaptability; applying the reactive system in multiple intelligent spaces is difficult. Imagine two spaces, one serving as an intelligent conference room and the other as an intelligent living room. While these two may share some reactions, many of their reactions will vary. The conference room would require the ability to react to someone giving a presentation; the living room, the ability to react to someone hosting a party. They both would require the ability to turn on the lights when it is empty and someone enters.

In creating these spaces, it would be helpful to have a separation between the system and the reactions, so that the same reactive system could be implemented in multiple rooms, supporting different activities. Developers would then be able to create a "presentation" reaction module for the conference room, a "hosting" reaction module for the living room, and an "empty room" reaction module for both. Our system needs a level of *abstraction*, bifurcating reactivity into an infrastructure and a set of reactions.

These limitations of the existing system used for Intelligent Room reactivity make evident the need for a better reactive behavioral system. In particular, this system should possess the following characteristics: context-awareness, modularity, conflict-resolution, and abstraction.

2.2 Sources of Inspiration

Our examination of the previous reactive system in the Intelligent Room brings to light several problems, highlighting four desiderata for a better system. For help in discovering how to endow a system with these characteristics, I look towards successful reactive systems in Mobile Robots and Self-Configuring Space Systems for inspiration. While the details of these systems are specific to their own areas, their design choices and overall thought processes are very valuable.

2.2.1 Mobile Robots

If we view the Intelligent Room as an "immobile robot," the similarities in creating a reactive robot and a reactive room become clear. The Intelligent Room, like a mobile robot, is subject to external influences to which it must react. However, as size and power consumption (among other differences) are important in the design of a mobile robot (but not in that of the Intelligent Room), this analogy can only be partially extended. Nevertheless, we can still learn from this research.

In the area of Mobile Robots, we look at *subsumption*, a layered architecture developed by Brooks [3]. The subsumption architecture allows a robot to safely meander through an unfamiliar surrounding. The overall control system comprises of multiple *layers of competence*, where a layer of competence is "an informal specification of a desired class of behaviors for a robot over all environments it will encounter." In addition, a higher level can override, or *subsume*, a lower level by inhibiting its outputs. By default, however, lower levels will continue to respond even when higher levels are added.

By segmenting behaviors into different layers, with higher priority layers overriding those with lower priority, the subsumption approach creates a mobile robot with a robust and complex behavioral pattern. In other words, a robot's behavior emerges from the interaction of simple layers, an idea similar to Minsky's "Society of Mind" theory. Like Minsky, Brooks believes that complex behavior can be created out of simple components.

An approach similar to subsumption would provide the Intelligent Room with a robust and complex reactive behavioral system. Segmenting room reactivity into modular components would create a forced organization of behaviors. A segmentation based on activity would provide the guidelines for this organization, while helping to create the activity-centric reactive modules needed to implement the system in different intelligent spaces. Subsuming layers of reactive behaviors would help integrate these reactive modules into a coherent reactive behavior. In ReBa, we adopt the idea of using a composition of simple reactive modules to create an overall complex reactive behavior.

2.2.2 Immobile Robots in Space

The "immobile robot" analogy also makes it easy to compare reactive system research in the Intelligent Room to that in Space Systems. Researchers in the field of Self-Configuring Space Systems already consider their control systems as immobile robots. In [44], Williams and Nayak coin the term "immobile robot," first in the context of ubiquitous computing. They then show that all robots have an immobile, autonomic component: "the focus of the attention of immobile robots is directed inward, toward maintaining their internal structure, as opposed to traditional robots, whose focus is on exploring and manipulating their external environment."

These control systems are in charge of goal-directed planning for a spacecraft. A spacecraft system requires reactive behaviors that help withstand the hostile space environment. It needs to react to temperature changes, collisions from small particles, malfunctioning internal components, and other similar problems. When a problem arises, human intervention is fairly impossible due to the inaccessibility of space; a self-monitoring system is necessary. Fault detection and re-planning are important aspects of these systems.

Livingstone, one such control system, uses a goal-directed, reactive, modular, and state aware method to address this problem [45]. The system can adapt its actions by understanding in which state it currently is. For example, say that a fault F is more likely in state A than in B. In accounting for fault F, the system would then take into account whether it is in state A or B.

Like these space systems, Intelligent Room reactivity is focused on maintaining the state of its components, though based on how it is affected by human users instead of environmental conditions or malfunctioning components. In Livingstone, the state-transitions model forces the control system to consider the context of the current failure. Applied to the Intelligent Room, this idea suggests using a system of states and transitions for maintaining the context needed in reacting to a user's actions. Unlike Livingstone, however, the ReBa system will react using a set of simple reactive modules rather than a model of the environment.

2.3 Understanding Context

A common theme emerges from the past two sections. The problems crippling the Mess-based reactive system and the ideas characterizing Livingstone share an emphasis on context maintenance. In other words, our analysis has shown that the room needs to situate a user action within the context of what the user is doing; the room needs to be context-aware. In this section I discuss the meaning of context, examine previous work with context in pervasive computing and intelligent environments, and define *activity-context*, the type of context that I argue our reactive system needs to maintain.

2.3.1 Context and Context-Awareness

Within ubiquitous and pervasive computing, there has been a large amount of research in understanding and using context, resulting in a variety of definitions. A thorough survey of this research can be found in [13], where Dey et al. settle on the following definition: "any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application...." We use this meaning of context in our research.

The term "context-aware" computing was first presented in the area of mobile computing [38]. In [37], the same authors elaborate on their idea, explaining that "context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as changes to such things over time." An Intelligent Environment, with its multi-modal sensing abilities, should be able to adapt to more user information;

while this work is relevant, this definition is insufficient.

In [13], the authors also examine the requirements of a context-aware application in pervasive computing, proposing three functions that such an application can perform:

1. Presenting information and services
2. Automatically executing a service
3. Attaching context information for later retrieval

Our reactive system should carry out the first two functions: it should present relevant information and services and execute relevant tasks based on what the user is doing. It is unclear whether the system needs to store contextual information; for the sake of simplicity, our reactive system will not be perform this task.

Dey et al. also argue that the acquisition of context should be separated from its use. By creating this layer of abstraction, we can make it easier to develop context-aware applications: applications would not have to worry about the sensors, and vice-versa. For this reason, our reactive system will not collect perceptual information, but instead will rely on other systems to provide this knowledge in the form of events.

2.3.2 Context Work in Intelligent Environments

Various research in Intelligent Environments is starting to examine how context should be used, and is building an infrastructure to support it. Microsoft Research's *EasyLiving* project presents the idea of a *context-aware intelligent system* (CAIE): "a space in which a ubiquitous computing system has contextual awareness of its users and the ability to maintain consistent, coherent interaction across a number of heterogenous smart devices" [39].

Through their work, we can understand better how a context-aware reactive system would affect a user's experience in an IE. As they explain, "CAIEs allow physical objects to become additional UI devices." For example, the Intelligent Room could use the movement of a door to learn of a user entering or exiting; it could use the lights to tell the user that it has recognized the entrance or exit.

Prekop and Burnett also provide a good explanation of what a ubiquitous computing environment can gain from context-awareness: "the ability to adapt the services or information it provides by implicitly deriving the user's needs from the context [surrounding the user]" [35]. A context-aware Intelligent Room would better serve its users by understanding user needs.

In developing their applications, Prekop and Burnett organize their information along a model of context they call, *activity-centric context.* They loosely define this term as information surrounding "the performance of an activity by an agent," where an agent is a person, a group of people, or a machine, and an activity is "a description of something being done by the agent." This model is designed to represent the tasks being carried out by the user. We want our reactive system to understand what the user is currently doing; our context model should be similar to theirs.

2.3.3 Using Context in our reactive system

Contextual information in the room needs to include the number, location, and current state of persons and objects in the space. For the purpose of room reactivity, I argue that this information needs to be organized by user activity. That is, our reactive system needs to maintain *activity-context*, a term which is derived from Prekop and Burnett's work and adapted for our work in IEs. We define *activity-context* as information about the current ongoing task, or activity, performed in the environment by the current user(s).

Our system also needs to take into account the order of the tasks started by users. When a user is carrying out multiple tasks at the same time, he is often carrying them inside each other. For example, a user may start a movie after entering the room; or, he may enter the room, start a meeting with other users, and then show the movie (or video clip). Our reactive system should understand that in the first example, the movie activity was started within an occupied room, while in the second, it was started within a meeting, and the meeting was started within the occupied room.

Other research has shown that activity-based computing can decompose user actions into its significant components: "Activity theory divides human behavior into a hierarchy" [5]. Our system can use its model of activity-context to break down user activity into the modular components that our previous analysis has shown as necessary. We can encode our room reactions at the level of individual activities, and then structure these components to reflect the activity hierarchy.

2.4 Lessons Learned

The first part of this chapter showed why the previous reactive system for the Intelligent Room, based on Mess, was inadequate. With its multiple shortcomings, it was in a mess, so to speak. A diagnosis of its problems teaches us that a better reactive system would be context-aware, modular, conflict-resolvable, and abstractive.

But how to endow a reactive system with these qualities? Or in general, how to endow the Intelligent Room with complex behavior? To answer these questions, this chapter then examined successful reactive systems in other research areas. From subsumption in Mobile Robots, we learn that complex behavior can emerge from simple components, and that a layering structure can be used to integrate the components. From Livingstone in Space Systems, we reinforce our trust in context, and learn that a system of states and transitions can be used to represent context. But what is context? What sort of knowledge should be embedded in this context? To grapple with these questions, this chapter studied previous work with context in IEs, identifying the importance and relevance of information about the ongoing activities in the room.

For the Intelligent Room to intelligently react to its users, it needs to understand human behavior. Activity theory allows us to break down human behavior into a hierarchical structure of simpler components. To generate room reactivity, we can encode reactions at the level of these components. With these activity-centric components, implementing a reactive system across different intelligent spaces also becomes easier. Subsumption suggests how to then put these components back together into a cohesive room behavior.

In this manner, I establish the need for the following four requirements in a reactive behavioral system for the Intelligent Room:

1. *Context-Awareness* - The reactive system should be aware of the activity-context of user actions

2. *Modularity* - Reactions should be encoded in modular components at the activity level

3. *Conflict-Resolution* - The reactive system should use a layering architecture to resolve conflicts among reactions

4. *Abstraction* - Intelligent Room reactivity should be divided into a generic infrastructure and a series of modular reactive components

The next chapter examines what decisions need to be made in the design of such a system, in order to implement these ideas.

In preparing for battle I have always found that plans are useless, but planning is indispensable.

– Dwight D. Eisenhower

Chapter 3

Design Decisions

This thesis so far has studied the problem of creating a reactive behavioral system for the Intelligent Room, identifying difficulties and the considerations that a solution will have to make. These next two chapters present a solution, starting from the design, continuing with an implementation, and ending with an evaluation. This chapter begins with the design decisions.

Our analysis of this problem has convinced us to pursue a context-aware, modular, conflict-resolvable, and abstractive reactive system. Our discussion in the first chapter adds the principles of user-centricity and evolvability. This chapter examines these ideas further, proposing a set of five requirements that our system should have. Each of these requirements is described in one of the following five sections (figure 3-1).

3.1 Modular Reactive Components

Marvin Minsky and Rod Brooks teach us that intelligent behavior can be broken down into simpler components [28]. To endow a space with intelligent reactive behavior, we need to begin by identifying what these components should be. In the last chapter, I argued that reactions should be encoded at the activity level. In this section, I describe how this should be done.

Each individual Intelligent Room reaction should be grouped with other reactions that are relevant within the same activity. I call each of these collections a *behavior bundle*, a group of reactions that correspond to the same activity. For example, imagine that the current activity in the room is a formal meeting. Possible room reactions in this context

Design Characteristics \ Design Decisions	Reactive Modular Components	Layering Architecture	Coordinating Glue	Customizability	Extensibility
Context Awareness	x	x			
Conflict Resolution		x	x		
Adaptability	x		x		x
User-Centricity				x	x
Evolvability					x

Figure 3-1: How each design characteristic is addressed by design decisions

could be to display the slides on the main wall, to darken the room when the movie starts, or even to send all incoming phone calls to the room to voice-mail. These reactions, then, would all be grouped into a behavior bundle corresponding to a formal meeting. See figure 3-2 for examples of room reactions, and figure 3-3 for examples of reaction organized by activity.

Another advantage of grouping reactions is that it enforces a systematic organization. One of the problems of the previous reactive system is that it relied on developers to understand where to add and group Mess rules. With activity-centric behavior bundles, developers understand that to add a new reaction, they must identify the activity where that reaction is relevant, and then add it in the corresponding bundle (or, if necessary, create a new bundle for that activity). To modify an existing reaction, a developer must follow a similar process.

The idea of modular reactive components also makes it easy to create a reactive system for multiple intelligent spaces. Two spaces, say a conference room and a living room, will share some reactions, but not all. Both spaces may react similarly when a user enters them when they are empty, but the conference room has a collection of reactions for a meeting,

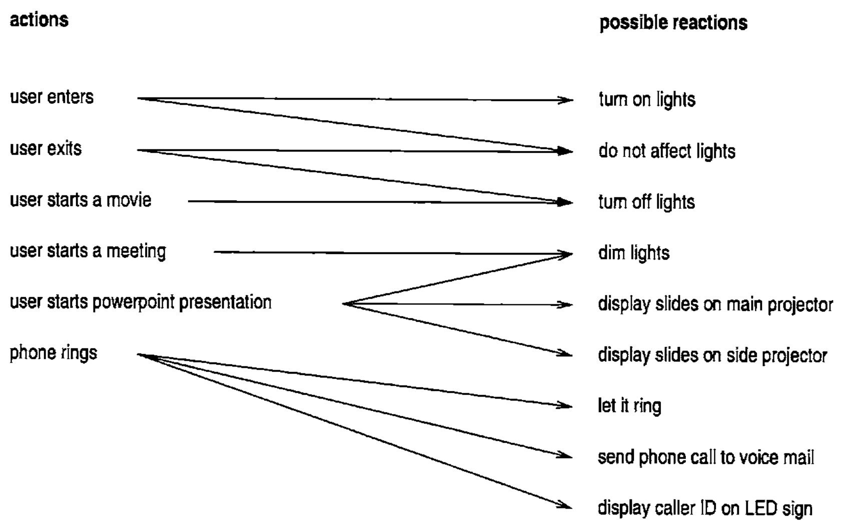

Figure 3-2: Examples of Intelligent Room reactions

and the living room a collection for hosting a party. Users of each space can decide which reactions they want their room to exhibit by selecting behavior bundles corresponding to the activities that they want to support. Developers, then, in this example need only write three bundles (instead of two for each room), avoiding code redundancy. For this adaptability to occur, we need to protect the autonomy of these bundles; a behavior bundle in itself should be room-independent.

3.2 A Layering Architecture

To create a reactive system from these components, we need a way to put them together; we need an architecture to integrate all the individual behavior bundles. This architecture also needs a way to resolve conflicts by arbitrating among competing behavior bundles. This section describes a layering structure, inspired by subsumption, for this purpose.

An intelligent space can support a variety of activities. Similarly, a variety of bundles may be installed in a space. At any given time, however, only a subset of these activities is being performed; only the reactions relevant to this set of activities should be exhibited by the room. We need to differentiate between those bundles that correspond to this subset of activities from the ones that are not applicable. When the activity corresponding to a bundle is being performed, I say that the behavior bundle is *active* and I say that it gets

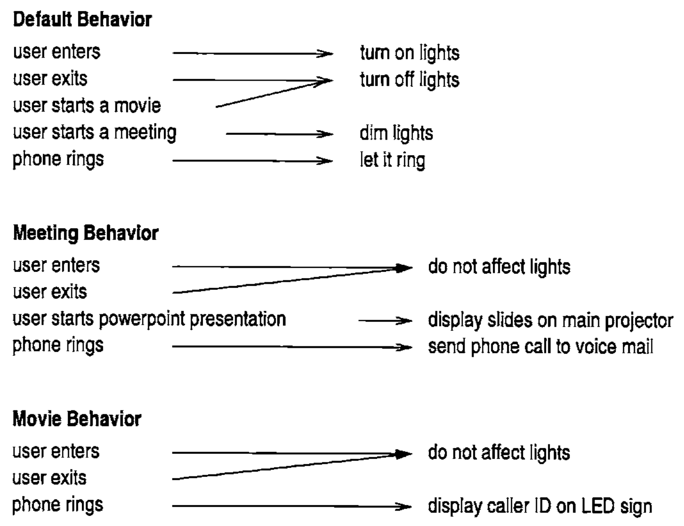

Figure 3-3: Organizing reactions by activity

activated when its activity is started.

In this way, I establish that at any given time, a collection of behavior bundles are active. What happens when two reactions, each contained in an active bundle, conflict with each other? Imagine that one bundle turns the lights on when a user enters, while another makes sure that the lights are not affected. When both of these bundles are active, how should the room prioritize one over the other?

Our system should have a layering architecture for active behavior bundles to address this concern. Imagine an empty room where only one bundle, the "Empty" behavior bundle, is active. When a user enters, the "Occupied" bundle is activated, and if it has a conflict with Empty, it layers itself on top of it. In other words, as a bundle is activated, it places itself "on top" of the existing active behavior bundles with which it has a conflict (figure 3-4). For any conflict between two bundles, the bundle in the higher layer overrides the lower one.

This layering technique of conflict resolution also helps to maintain activity-context. As explained earlier, an activity-context representation needs to take into account the order of activities. It needs to understand which activities were started within the context of others. Our architecture preserves order through layering: if a behavior bundle is in a higher layer than another, at that moment its context is a sub-context of the other.

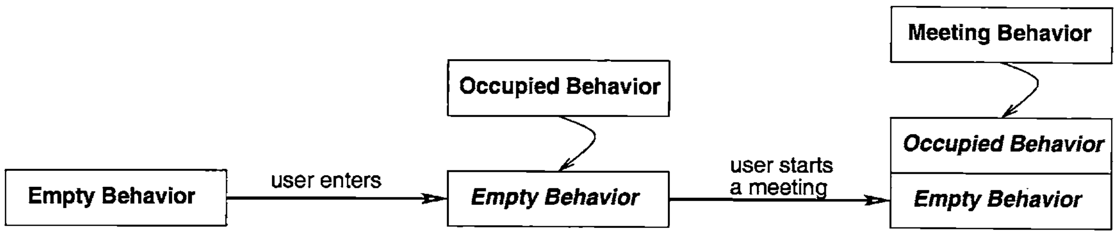

Figure 3-4: Simple layering example

3.3 A Coordinating Glue

Our independent behavior bundles need to discover and communicate with each other; our system needs a coordinating infrastructure to provide for this interaction, tying these bundles together. This infrastructure, or glue, needs to protect the autonomy of each behavior bundle, balancing the interdependence with the independence.

On the one hand, we want to sustain the modularity of these bundles, allowing users to select which modules they want to install in their rooms. On the other hand, we want our bundles to resolve their conflicts; we want each bundle to discover other bundles with which it has conflicts. In each intelligent space, users will decide which activities to support, and will then install the corresponding behavior bundles. For each space, the installed bundles will need to discover each other, forming relationships if they have conflicts.

In addition, when the reactive system receives an event from a perceptual system, the coordinating glue needs to provide a communication backbone for bundles to pass events to each other. This message passing model needs to preserve the layering architecture.

3.4 Customizability through User Feedback

These last three requirements are concerned with the functionality of the reactive system. These last two focus on its usability. This section starts with a discussion of the customizability of the system.

Although the first three design requirements use activity-context to mirror user expectations, the system may still fall short. No matter how good the understanding of human behavior, it is impossible to predict the whims and fancies of each individual user, as well as every improbable exception. For example, Bob may decide that he wants the drapes open and the lights on all the time. Alice may like the lights turned off for a formal meeting, but only dimmed for more casual ones. A room that does not understand these changing needs, but instead turns on the lights when it should not, would irritate users.

To avoid this problem the reactive behavioral system should be customizable by the end users. Ideally, the system would learn from user feedback. If a user consistently turns off the lights during a specific activity, then the behavior bundle corresponding to that activity should incorporate that feedback.

3.5 Extensibility through Programmability

Our system would also be more usable if it were easily extensible. Our system should provide a room with more than a predefined set of intelligent, context-aware reactions. It should also allow users to create more sets of reactions that would use the same system infrastructure; it should allow users to extend the system by creating new behavior bundles to incorporate more activities and contexts.

Making the design of new behaviors easy for all users is a difficult task. But making it easy for those users comfortable with programming, for example, other Intelligent Room developers, is much easier. With this mind, learning how to program new behaviors should not be much more difficult than writing basic Intelligent Room components.

3.6 Summary

In this chapter, I discussed five features for our reactive behavioral system:

- Modular Reactive Components: Reactions should be grouped by activity into modules called *behavior bundles.*

- A Layering Architecture: To integrate the reactions of these behaviors, avoiding conflicts, the system should use a *layering* architecture.

- A Coordinating Glue: The system needs a coordinating infrastructure to allow behavior bundles to *discover* and *communicate* with each other.

- Customizability: Room reactivity should be *customizable* by the end users.

- Extensibility: The system should make the design of new bundles easy for Intelligent Room developers.

The next chapter describes an implementation of these principles.

If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them.

– Henry David Thoreau

Chapter 4

The *ReBa* System

We have decided to ask the following requirements of our system: modular reactive components, a layering architecture for arbitration and discovery, a coordinating glue for communication and message passing, customizability through user feedback, and extensibility through programming. Using these requirements, we have designed *ReBa*, an agent-based reactive behavioral system for the Intelligent Room (figure 4-1).

ReBa features:

- *Behavior agents* for modularity
- *Relationships* between Behavior agents for arbitration
- A *Behavioral Network Model* for discovery
- A *Behavior Coordinator* to coordinate message passing among all Behavior agents
- A *Behavioral Tree* for message passing among active Behavior agents

ReBa is implemented in the Metaglue multi-agent system, which is built on top of Java. ReBa uses Metaglue as the fundamental communication backbone, and uses the inheritance and reflection qualities of Java for extensibility.

In addition, ReBa makes use of *macros* for customizability. Macros are being independently developed at the Intelligent Room project.

	Implementation							
Design Decisions	Java's Inheritance and Reflection	Metaglue	Behavior Agents	Behavioral Relationships	Behavior Coordinator	Behavioral Network Model	Active Behavioral Tree	Macros
Reactive Modular Components		x	x					
Layering Architecture				x				
Coordinating Glue		x			x	x	x	
Customizability								x
Extensibility	x							

Figure 4-1: How each design decision is implemented

4.1 Modularity

Intelligent Room reactions are organized into behavior bundles, which are autonomous activity-centric reactive components. Creating bundles around activities allows us to begin representing activity-contextual knowledge. The modularity of these components, along with the layers of abstraction separating away the perceptual systems and the resource manager, helps us create an adaptable reactive system that can be implemented in various intelligent spaces.

4.1.1 Behavior Bundles

As described in the previous chapter, a behavior bundle is a collection of Intelligent Room reactions that pertain to the same activity; each bundle corresponds to an activity. Examples of activities (with their respective bundles) are watching a movie (and the Movie behavior bundle), and participating in a meeting (and the Meeting behavior bundle). A bundle can also correspond to a sub-activity. A sub-activity is an ongoing task that is an explicit type of another activity. For example, a presentation and a brainstorming session are both types of a meeting. The Presentation and Brainstorming bundles are types of the

```
public void actionLights(Secret secret) {
  if (secret.isA("perception.cloud.door open")) {
    try {
      lights.turnOn();
      log("DEBUG", "actionLights");
    } catch(Exception e) {
      log("ERROR", "Error during Lights Action: " + e);}}}
```

Figure 4-2: Sample action method

Meeting bundle.

The reactions contained in a bundle specify outputs for given conditional inputs: given specific perceptual events, they respond by changing the state of the room and its devices. For understandability and ease of maintenance, these reactions are grouped if they provide similar services. These groupings are called *actions*. A set of reactions controlling the illumination in the room, for example, would be organized into a "lights" action, while another set welcoming the user constitutes a "greeting" action. A behavior is thus a collection of actions (figure 4-3).

4.1.2 Behavior agents

Our behavior bundles fit nicely with the autonomous agent model of Metaglue. As a result, each bundle is implemented as a Metaglue agent, called a *Behavior agent*. For example, the Movie, Meeting, Brainstorming, and Presentation behavior bundles are implemented as *Movie, Meeting, Brainstorming, and Presentation Behavior agents.*

Actions in a Behavior agent are implemented as Java methods. Each Action method contains a series of conditional clauses that provide reactivity (figure 4-2). When a clause receives a perceptual event, it obtains a room resource (through the resource manager), and makes changes to the state of the room. For example, when the "lights" action receives an event stating that someone has entered the room, one of its clauses obtains control of the lighting devices and turns them on.

To simplify the design of a new behavior bundle, providing extensibility, ReBa takes advantage of the inheritance and reflection qualities of Java. All Behavior agents inherit from a Metaglue class called the *BehaviorAgent superclass.* Using reflection, most of the computation of a behavior is performed by the superclass; the superclass abstracts away the

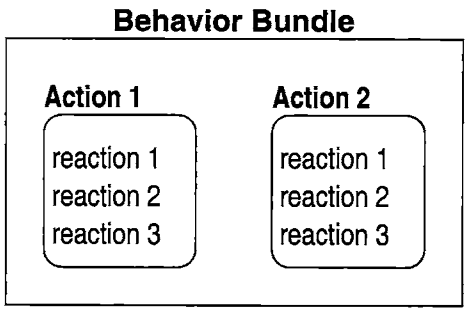

Figure 4-3: Generic behavior bundle with reactions organized into actions

computation common to all Behavior agents. This computation includes maintaining the Action methods, listening for an event, and passing an event to the appropriate actions. A template for making a new Behavior agent can be found in appendix A, and sample Behavior agents can be found in appendix B. It is difficult to analyze how easy the BehaviorAgent superclass makes the design of new behaviors.

Each Behavior agent also contains a *BehaviorID*, which is an identifier unique to that bundle. The BehaviorID includes the agent's *AgentID*, which is its unique Metaglue identifier.

4.1.3 Behavior agent States

An intelligent space allows its users to perform a variety of activities: watching a movie, running a meeting, giving a presentation, etc. At any point in time, however, only a subset of these activities is being performed. Users may watch a movie; or run a meeting and watch a video clip; or run a meeting and give a presentation; etc. Similarly, a variety of behavior bundles may be installed within a space, but at any given time only a subset are *active*.

To help recognize whether a bundle is active or non-active, each Behavior agent is endowed with a state. Each bundle maintains whether it is active or non-active. It activates itself when a user starts its corresponding activity, and deactivates itself when the user stops.

There are other activation models. [27] presents an action selection algorithm for activating agents. These agents, like Brooks' robots, are composed of a hierarchy of competence modules. Like our bundles, these modules activate and inhibit each other through various relationships: successor links, predecessor links, and conflictor links. Unlike our discrete state model, these modules possess a continuous quantity of *activation energy.* A module receives energy when one of its predecessors receives energy or is activated; it loses energy when a predecessor loses energy or is deactivated.

The continuous model is necessary for the system in [27] because its modules sense and react. In the Intelligent Room, the notion of energy levels is encoded in our perceptual systems. Our activation model needs discrete logic for intelligibility; the system needs to know whether a behavior bundle is active or not, not a level of activation.

4.1.4 Behavior Speech agent

The Behavior Speech agent was created to abstract away the issues related to interacting with the Metaglue speech infrastructure. As a result, each Behavior agent communicates with the Behavior Speech agent for registering grammars and outputting spoken phrases.

4.2 Arbitration and Discovery

As autonomous components, behavior bundles are entrusted with the task of assembling themselves into a layering architecture. For this to happen, bundles need to discover each other and arbitrate amongst themselves. Behavior agents form relationships with each other to prioritize themselves. Conflicts are resolved by the bundle with the higher priority *overriding* the bundle with the lower one. A stronger form of this overriding relationship is a *dependency*, where one bundle corresponds to a sub-activity of the other. For these Behavior agents to discover each other and form these relationships, they use the *Behavioral Network Model.*

4.2.1 Behavior Bundle Relationships

The two behavior bundle relationships, *overriding* and *depending*, allow active bundles to layer themselves. As stated earlier, when a user starts an activity, the bundle corresponding to that activity activates itself. In this way, bundles activate themselves based on the order of user actions. If an activity is performed within another, the bundle corresponding to the one started second activates on top of the first, *overriding* it.

Overriding occurs among active bundles on the level of their actions. If a behavior

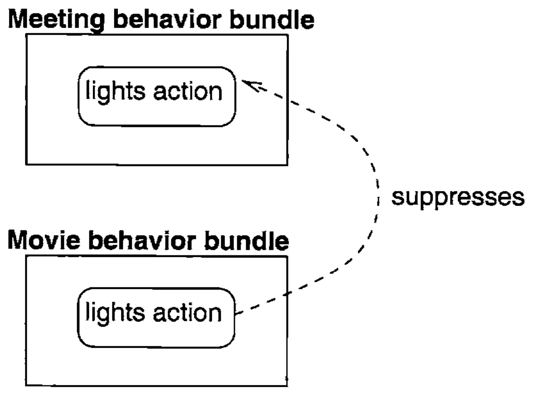

Figure 4-4: Movie overriding Meeting: Basic overriding model between two behavior bundles

bundle overrides another, then it tells the other that one of its actions is suppressing one of the actions of the other. For clarity in the overriding process, similar actions in different bundles are given the same name. For example, if the Movie behavior bundle overrides the Meeting bundle by taking control of the lights, then the Movie "lights" action suppresses the Meeting "lights" action (figure 4-4). This action remains suppressed until Movie deactivates itself.

Note that a behavior bundle is overridden if it has at least one suppressed action. Also, a bundle may be overridden multiple times; one bundle may override another that is already overridden (figure 4-5).

A *dependency* is a stronger form of overriding that helps to preserve the relationship between contexts and sub-contexts. Dependencies were created to help identify the behavior bundles, among all non-active bundles, that should be allowed to activate themselves. In other words, given the current context, dependencies were created to pass events to the non-active bundles who are permitted to activate themselves. A behavior bundle is allowed to activate itself if at least one of the bundles that it *depends* on is active.

For example, if no one is in the room, then regardless of the event that the Meeting bundle receives, it should not be allowed to activate itself: a meeting would not occur in an empty room. Similarly, if a meeting is not occurring, then regardless of what the Presentation and Brainstorming bundles think, they should not be allowed to activate themselves. In these examples, Meeting *depends* on the Occupied bundle; it is allowed to

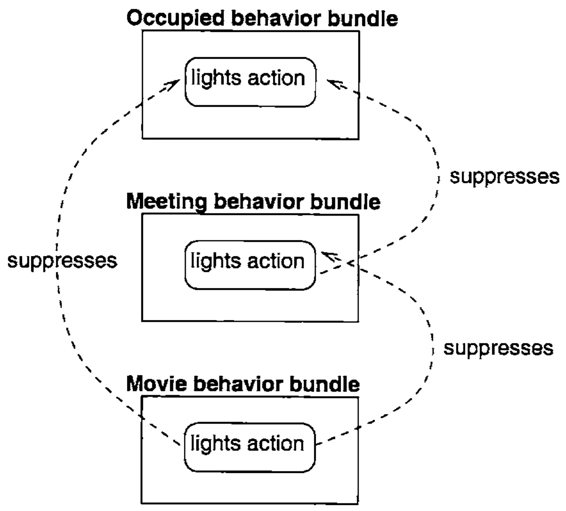

Figure 4-5: Occupied overridden by Movie and Meeting: One bundle being overridden by multiple bundles

activate itself if the room is occupied. Similarly, Presentation and Brainstorming *depend* on Meeting; they are allowed to activate themselves if a meeting is being performed (figure 4-6). Conversely, if the room is not occupied, then neither Meeting, Presentation, nor Brainstorming are allowed to activate themselves, and the system does not pass them any events.

When a non-active bundle has an active dependency, and is thus allowed to receive events and activate itself based on those events, we say that it is just *listening*. A non-active bundle that is not listening is simply *inactive*; inactive behavior bundles depend on other bundles that are either listening or also inactive. In this manner, dependencies are used to prevent certain non-active behavior bundles from activating themselves.

When a behavior bundle does activate itself, it layers itself onto the active bundles that it depends on, overriding them.

4.2.2 Behavioral Network Model

When a behavior bundle is installed into a space, it is ignorant about others in the same space. It uses the information encoded in the *Behavioral Network Model* to discover the others and form relationships.

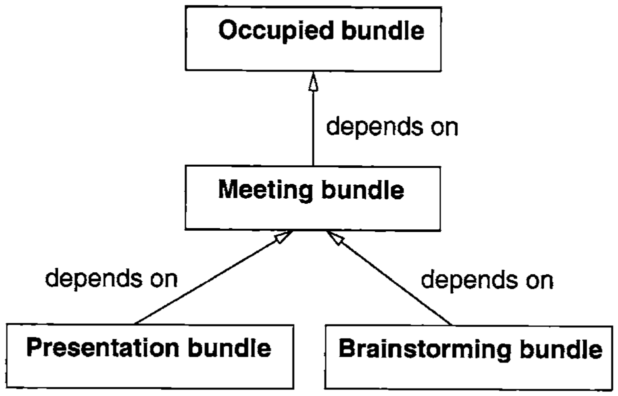

Figure 4-6: Dependency relationships between behavior bundles

The Behavioral Network Model (figure 4-7) describes the overriding and depending relationships between all the behavior bundles installed in that space. Each overriding relationship is represented as a connection from the overriding action in one bundle to the suppressed action in the other. Each dependency is represented as a connection from one bundle to the bundle that it depends on.

We rely on a user to identify these relationships and make these connection in the network model. Each behavior bundle is packaged with an API specifying its properties: its corresponding activity, its actions, etc. The user makes use of a bundle's API to install the bundle into the space. Typically, this would only be required when a bundle is installed into the room.

We choose this system over other methods of discovery for various reasons. One way to create relationships between bundles is to hard code them into the bundles themselves. This method is inadequate as it prevents our reactive system from being adaptable: it forces a bundle to be designed for the other bundles in a particular space. Another method would be to describe a behavior bundle using a meta-language that was parsable by other bundles. Bundles would then use these descriptions to discover each other. This method is more complex than our current one; for the sake of simplicity, we choose not to use it for ReBa.

For simplicity, we also choose programming as the way to input information in the behavioral network model. More complex solutions would provide more natural input mecha-

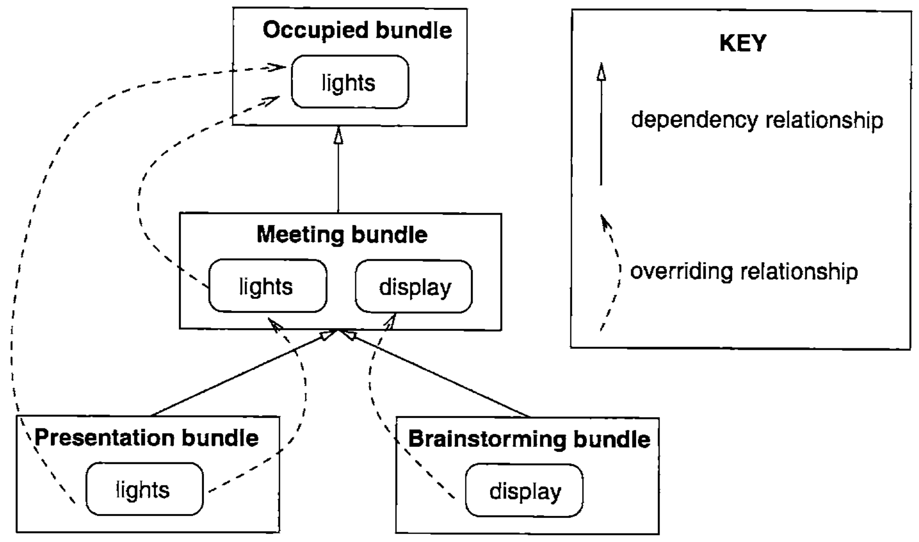

Figure 4-7: Behavioral Network Model: Overriding and dependency relationships between behavior bundles

nisms. For example, a solution could involve a version of *Ligature* [17], a prototype graphical interface that allows a user to observe and modify an intelligent space's devices through sketching.

4.3 Message Passing Infrastructure

In the Intelligent Room, we have separated our reactive system from the acquisition of perceptual information. The room's perceptual systems collect information, and broadcast it across the Metaglue communication backbone in form of events. Our reactive system needs to listen for these events and pass them along to the appropriate behavior bundles. In addition, active bundles need a method of passing events amongst themselves, in accordance to the layering relationships. ReBa has a *Behavior Coordinator* to pass events from the perceptual system to the behavior bundles. Active behavior bundles form the *Active Behavioral Tree* to pass these events through the layering architecture.

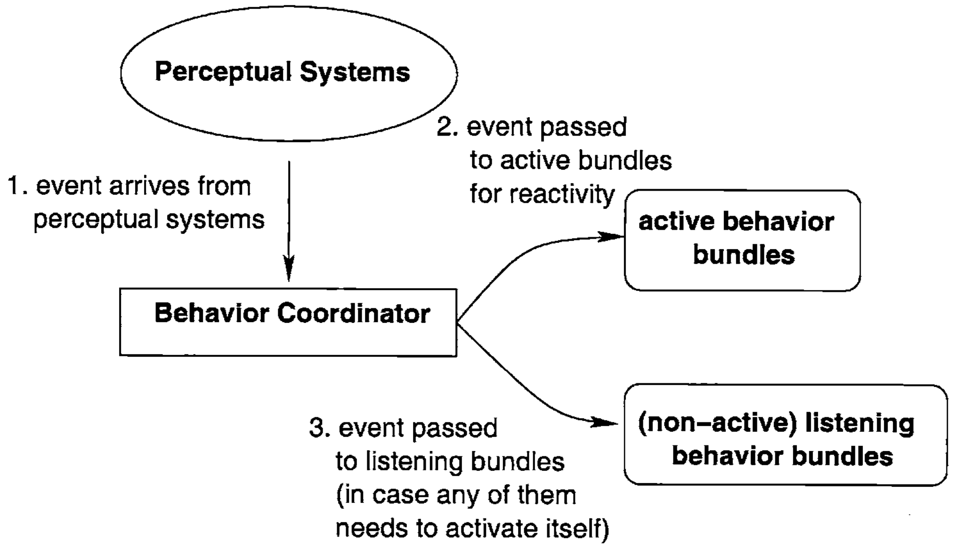
Perceptual Systems
1. event arrives from perceptual systems
Behavior Coordinator
2. event passed to active bundles for reactivity
active behavior bundles
3. event passed to listening bundles (in case any of them needs to activate itself)
(non-active) listening behavior bundles

Figure 4-8: Message passing model from the perceptual systems to the bundles, via the coordinator

4.3.1 Behavior Coordinator

The *Behavior Coordinator* is an agent that acts as the interface between the behavior bundles and the room's perceptual systems. Each intelligent space is assigned a Behavior Coordinator to manage the bundles installed within that space. When a behavior bundle is installed, it registers itself with the coordinator. To register, a bundle transmits its *BehaviorID* to the coordinator, allowing the coordinator to communicate with the bundle in the future. During registration, the bundle also tells the coordinator which events it wants the coordinator to listen for. In this way, the coordinator does not need to listen to every possible event; it can limit itself to the events that its bundles want to listen to.

When the coordinator receives one of these events, it passes it to the top layer of active bundles. After all the active bundles have responded to the event, the coordinator passes it to the non-active listening bundles, in case any of them need to activate themselves (figure 4-8).

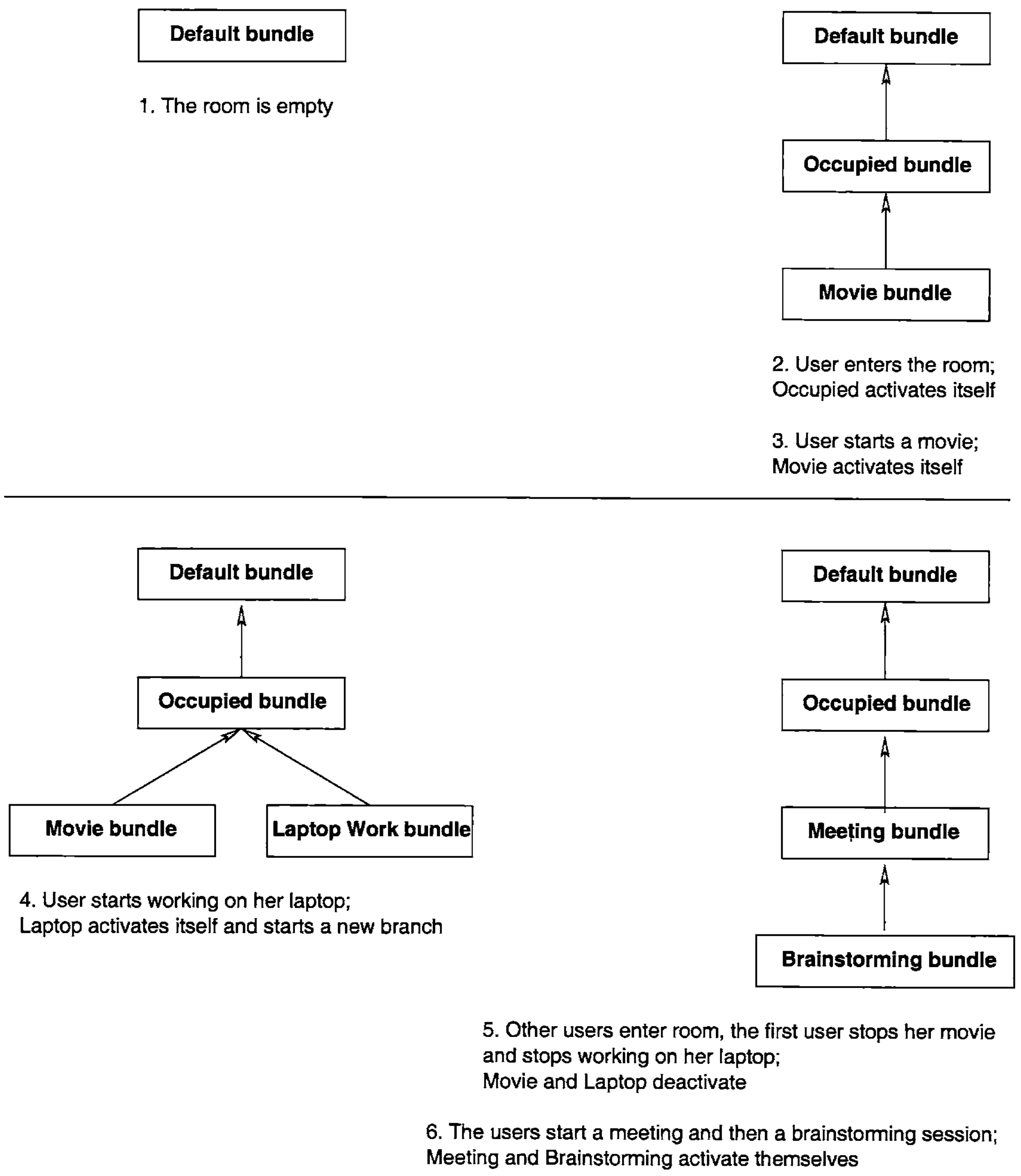
Default bundle
1. The room is empty
Default bundle
Occupied bundle
Movie bundle
2. User enters the room; Occupied activates itself
3. User starts a movie; Movie activates itself
Default bundle
Occupied bundle
Movie bundle
Laptop Work bundle
4. User starts working on her laptop; Laptop activates itself and starts a new branch
Default bundle
Occupied bundle
Meeting bundle
Brainstorming bundle
5. Other users enter room, the first user stops her movie and stops working on her laptop; Movie and Laptop deactivate
6. The users start a meeting and then a brainstorming session; Meeting and Brainstorming activate themselves

Figure 4-9: The creation of the active behavioral tree based on user actions. Events are passed up the tree along the arrows.

4.3.2 Active Behavioral Tree

When the Behavior Coordinator passes an event to the top layer of active bundles, the bundles handle the event, and then pass it down through the layers, according to the relationships that they have formed. This message passing architecture is called the *Active Behavioral Tree.* All active behavior bundles are nodes of this tree (figure 4-9).

We chose to use a tree structure to preserve the hierarchy of contexts. By maintaining activity-context, we are able to decompose user behavior into a collection of activities. Two activities being simultaneously performed in the same room can be either independent of each other (e.g., a user working on her laptop while watching a movie in the background), or sub-activities of each other (e.g., users participating in a brainstorming session as part of a meeting). We want our message passing structure to reflect separate contexts and sub-contexts, independent bundles and related bundles. A linear message passing model would not allow us to maintain this distinction. In our tree metaphor, separate contexts are different branches of the tree, while sub-contexts are in the same branch.

As nodes of the active behavioral tree, behavior bundles can be classified into two categories: *leaf active* bundles and *overridden active bundles.* The leaf bundles form the highest layer of behaviors; they are not overridden. Similarly, the overridden bundles form all the other layers underneath; an overridden behavior bundle is overridden by at least one other bundle.

The one behavior bundle common to all spaces, the *Default* bundle, is the root of this tree. The Default behavior bundle is always active, and does not contain any actions. The next bundle to activate itself is attached as a child node to the root node. This bundle is now "leaf" of the tree; it becomes a leaf active bundle, while the Default behavior becomes an overridden active bundle. Then, as each behavior bundle activates itself, it scans through the leaf nodes, finding a behavior on which it can *depend.* It latches on as a child node of that behavior, overriding it and any others with which it has conflicts. In this manner, after a series of activities are started in the space, an active behavioral tree is created.

The Behavior Coordinator maintains lists of the leaf active and overridden active bundles. When a bundle activates itself and becomes a leaf, it tells the coordinator; when that bundle is overridden and no longer a leaf, it again tells the coordinator. To preserve the autonomy of the behavior bundles, and reduce computation, the coordinator does not know the actual structure of the tree. When the coordinator wants to pass an event to the active bundles, it sends the event to the leaf nodes. Each leaf behavior, after passing the event to all its actions, forwards the event up the tree to its parents. Each parent waits until all its children have forwarded the event, and then passes the event to its non-suppressed actions. In this way, each event is passed up the active behavioral tree. When the root node, the Default behavior bundle, receives the event, the event has passed through the entire tree.

Despite the parallel structure of the tree branches, the message passing model is still linear. When an event reaches the leaf nodes, it is first handled by each leaf active bundle. Only after all the leaf bundles have handled the event is the next level allowed to handle it. For the sake of simplicity, this approach was implemented instead of a parallel message passing model.

4.4 Customizability

One of our design requirements states that room reactivity should be customizable by the end user; behavior bundles should adapt to user feedback. In ReBa, we make use of a simpler mechanism, called *macros*. Macros, which are being independently developed at the Intelligent Room Project, contain a straightforward series of room reactions. We allow the user to perform (or command) a series of room reactions, and then store them as a macro. The user then names this macro with a unique phrase, and is able to call it whenever he wants by issuing a verbal command. He can call a macro to undo or correct room behavior; or, he can use the macro to produce his own desired reactions from the room.

If the user is dissatisfied with ReBa's behavior, we also provide for an extreme form of customizability: we allow the user to turn off ReBa through spoken commands. For this mechanism, we created a behavior bundle called *Vacant*, which applies when the room is empty. Vacant depends on the Default behavior bundle; all other bundles depend on Vacant or on a bundle that depends on Vacant. Through speech, a user can activate or deactivate Vacant. If Vacant is not active, then no other behaviors can activate themselves, and ReBa is in effect turned off.

4.5 Future Work

The balance between autonomy and coordination in our system allows ReBa to be a coherent consolidation of modular components. This system benefits from the usual advantages of modularity: developers can create multiple behavior bundles in parallel, and users can upgrade specific bundles with only a slight disruption (through the behavioral network model) to the rest of the system.

After developing ReBa, we were able to evaluate the decisions we had made in our implementation, and point out areas for future work:

- A better discovery mechanism for behavior bundles; or, a better input mechanism for the behavioral network model
- Behavior bundles that learn from user feedback
- A user study to understand how to make Behavior agents easier to implement

The next chapter compares ReBa to other similar research.

Life is often compared to a marathon, but I think it is more like being a sprinter; long stretches of hard work punctuated by brief moments in which we are given the opportunity to perform at our best.

– Michael Johnson

Chapter 5

Comparison with Related Systems

The research field of Intelligent Environments is not very large; the amount of research in reactive systems for IEs is even smaller. Our reading of other literature has shown us that very few individuals have paid as much attention to developing a system that reacts to users as ours does. However, as indicated in chapter 2, there has been a good deal of work with context in pervasive computing environments. This section examines a few of these projects.

5.1 EasyLiving

The *EasyLiving* project [4] at Microsoft Research has recently begun to explore the idea of context-aware reactivity in smart environments. One of its goals is to implement a *context-aware intelligent system* (CAIE) [39]. In developing EasyLiving, they propose four main ways for a CAIE to use context:

1. Resolving references
2. Tailoring lists of options
3. Triggering automatic behaviors
4. Tagging information for later retrieval

In the descriptions of the third method, they identify the ability of the room to use context for reactivity. They define *automatic behaviors* as "actions that the computing system initiates in response to contextual information."

These researchers also suggest six ways to incorporate automatic behaviors into a CAIE:

- Programmed (by the user explicitly)
- Taught (via demonstration)
- Learned (by observation)
- Fixed (built into the system; i.e., programmed by the manufacturer)
- Installed (by acquiring "behavior software" from a developer)
- Shared (transferred from another environment)

ReBa makes use of almost all six methods. Behaviors can by *programmed* by the user in Metaglue. Behaviors are also modular, allowing them to be *fixed* into the system, *installed* by the users, or even *shared* from another space. Our development of *macros* is a step towards ReBa being *taught* by the user. Finally, while we have also identified the need for behaviors to be *learned*, ReBa does not implement any method to accomplish this task.

So far EasyLiving has not proposed effective methods to accomplish all these tasks. They highlight two main difficulties in designing automatic behaviors: avoiding conflicts and cycles. Through ReBa's layering architecture, we have a good solution for handling conflicts; and while we do not address cycles, the activity-based organization of reactions should help developers recognize cycles. Even if developers fail to encode conflicts or detect cycles, users still have the ability to use macros to correct room behavior. If necessary, users can even deactivate the Vacant behavior, thereby deactivating ReBa.

5.2 The Aware Home

The overall goal of the Aware Home [25] at the Georgia Institute of Technology is to build a home that is entirely a pervasive computing environment. To incorporate context into their applications, they have created a *Context Toolkit*. The Context Toolkit is an architecture to help developers write context-aware applications for a user's environment [12]. It focuses on collecting contextual information and making it available.

Their applications include an *In/Out Board*, which displays information about the current building occupants; a *context-aware mailing list*, which sends an e-mail message to members of the research group currently in the building; and an augmentation of the *Dynamic Ubiquitous Mobile Meeting BOard (DUMMBO)*, to help the recording of informal meetings [13]. Each of these applications responds to contextual information about its users. For example, the DUMMBO application starts its audio recording when users have gathered around the white-board. These reactions appear to be programmed into the system.

While the Context Toolkit provides a good abstraction for adding contextual information to applications, the reactivity of these applications has some of the same problems of the Mess implementation. It is not modular, conflict-resolvable, or customizable. Also, as they do not focus on multi-application systems, they do not worry about adaptability.

5.3 Other Systems

There are several other Intelligent Environment systems dealing with similar issues. The KidsRoom [2] at the MIT Media Laboratory used non-encumbering sensors to create an interactive, narrative environment for children. The Gaia Project [36, 6] at the University of Illinois at Urbana-Champaign is creating a component-based operating system for ubiquitous computing. The Interactive Workspaces Project at Stanford has proposed a conceptual framework for designing interactive computing environments [46], and has already implemented a service framework [34].

Again, these projects have not paid as much attention to developing a reactive system like ours.

The great thing in this world is not so much where we are,
but in what direction we are moving.
– Oliver Wendell Holmes

Chapter 6

Concluding Thoughts

6.1 Contributions

ReBa is a fully functional component of the Intelligent Room software infrastructure. It provides a way to make use of perceptual information, including information from cameras, microphones, and sensors, for creating context-aware room reactions.

In this thesis, I started with five guiding thoughts: context-awareness, conflict-resolution, adaptability, user-centricity, and evolvability. I believe that these five guidelines form a set of design principles for a reactive system in any Intelligent Environment.

To follow these guidelines, I decided to ask the following requirements of our reactive system: reactive modular components, a layering architecture, a coordinating glue, customizability, and extensibility. Our requirements, in turn, were satisfied in ReBa by components that served the following purposes: modularity, arbitration, discovery, message passing, and again, customizability and extensibility.

In designing our system, we analyzed the problem from functionality and usability perspectives, trying to ensure that our system would work and be used. As a result, our system is both modular and integrated, portable and stationary, complex and simple.

6.2 Ants and Environments: True complexity lies in the user

In "The Sciences of the Artificial" [40], Herbert Simon uses an ant crawling on a beach as an example of seemingly complex behavior. His ant moves towards a goal, perhaps towards its colony, planning its route with a greedy algorithm. It moves forward, and on encountering an obstacle, chooses the best path around it. In this way, its path becomes a crooked sequence of straight lines and irregular curves. Sketched onto paper, this path looks like the behavior of a fairly intelligent creature. Indeed, the path is intricate, yet the intricacy does not lie in the ant: the ant is just responding to objects in its way. The complexity lies in the nature of the beach. The ant's behavior simply reflects the nature of its environment.

Our intelligent space is like this ant. Turning lights on and off, displaying data on walls, responding to body language, etc., our room exhibits intelligence and appears complex, almost daedal. Yet the complexity is not in the room itself; the complexity lies in us. The room's intelligent behavior, though seeming complex, emerges from the interaction between its simple components; it is the interaction that is complex, rather than the room itself.

Isaac Newton once postulated that, "To every action there is always opposed an equal reaction." As we act in our Intelligent Environment, the environment reacts to us. We are the environment's environment, so to speak. To bring computing into the world of natural human communication, we just need to make sure that the intelligent environments we build perceive and reflect our world. This idea is promising, and systems such as ReBa provide some leavening optimism for future research.

Appendix A

Behavior agent Template

```
/*
 * BehaviorTemplate.java--
 * A Template for designing Behaviors for ReBa
 * version 1.02
 *
 * DO NOT COMPILE!!!
 * This java file is designed as a reference for the creation of other
 * behaviors.
 *
 * A few general notes about designing behaviors:
 * a) when I say that you need to write a certain line of code, where
 * (within the body of the same method) doesn't matter. i.e., order of
 * what gets called when within a method doesn't really matter
 *
 * b) if you ever want the behavior to say something, you can call:
 *      bevSay("ENTER PHRASE HERE");
 *    if you want the behavior to say something for debugging
 *    purposes, you can instead call:
 *      bevSay("DEBUG","ENTER PHRASE HERE"); [like the log(,); call]
 *    (note that the flag for setting DEBUG mode on or off is in the
 *     attribute database, under
 *    agentland.behavior.speechin.BehaviorSpeech debugSpeech
 */
package agentland.behavior;
// if behavior is not in agentland/behavior/, then you need to
// import agentland.behavior.*;
```

```
import metaglue.*;
import java.rmi.*;
// basic metaglue imports

import agentland.util.*;
// for Secret

public class BehaviorTemplateAgent extends BehaviorAgent
    implements BehaviorTemplate {

    // the behavior AGENT file (i.e., the class) must:
    //      a) extend agentland.behavior.BehaviorAgent
    //      b) implement the behavior INTERFACE
    //
    // the behavior INTERFACE must:
    //         extend agentland.behavior.Behavior

    public BehaviorTemplateAgent() throws RemoteException {
super();
// Must call parent's constructor for initialization:
// As most of the *brains* of an agent are in
// agentland.behavior, the failure to make this call will
// lead in the behavior not working... at all

/*** OTHER INITIALIZATION ***/
// PUT whatever other initialization you want in your
// constructor

/*** SPEECH ***/
// add the grammar for this agent
// e.g.,
// addGrammar("agentland.behavior.speechin.behaviorTemplate");

// dynamically update behavior speech agent by linking a
// grammar component with an event that should be fired on the
// observation of that grammar component

// i.e., if the grammar says:
// public <starttemp> = start the template {starttemp};
// then, below, you should say something like:
// addKeyNotePair("starttemp","behavior.template.start");

// and even further below (i.e., in the EVENTS section), you
// should say something like:
// addEvent("behavior.template.start, "activate");
// OR
// addEvent("behavior.template.start, "actionABCD");
// (see the next section for more info)

/*** EVENTS ***/
// below, add the EVENTS (i.e., notifications) that you want
// to listen for, followed by the name of an ACTION that
// listens for this notification. for multiple ACTIONS for the
// same EVENT, repeat this line as necessary

// e.g.,
// addEvent("this.is.a.notification","actionXYZ");
// addEvent("this.is.a.second.notification","actionPQR");
// addEvent("this.is.a.second.notification","actionABC");
// addEvent("perception.cloud.door open", "actionLights");

// ALSO, like I mentioned earlier:
// addEvent("behavior.template.start", "activate");
// IN WHICH CASE, you are taking a spoken utterance recognized
// by the grammar, firing a notification based on the grammar
// component, listening for that notification, and then
// sending that notification to the particular action(s) that
// is/are registered for it

// NOTE that you need at least the two following EVENT lines,
// to establish which event(s) activate and deactivate this
// behavior:
// addEvent("this.is.a.notification","activate");
// addEvent("this.is.another.notification","actionDeactivate");
    }

    /*** ACTIONS ***/
    public void activate(Secret secret) {
// this action is called to activate a behavior,
// i.e., transform from LISTENING to ACTIVE mode
// AND SO, it needs to be in your behavior

makeActive();
// NEED to call this method. it tells the bev coord that
// you're now active
// DON'T FORGET to put what you want to happen!!
    }

    public void actionDeactivate(Secret secret) {
// this action is called to deactivate a behavior, i.e.,
// transform from ACTIVE to LISTENING or perhaps even INACTIVE
// AND SO, it needs to be in your behavior

makeListening();
// NEED to call this method. it tells the bev coord that
// you're (at least) listening now
// DON'T FORGET to put what you want to happen!!
    }

    // put any other actions you have designed below
    // make sure that their names start with "action-"
    // i.e., "actionXYZ, actionABCD, actionASDF"

    // e.g., actionLights
    public void actionLights(Secret secret) {
if (secret.isA("perception.cloud.door open")) {
    try {
     bevSay("Lights are On!");
log("DEBUG", "actionLights");
    } catch(Exception e) {
log("ERROR", "Error during Lights Action: " + e);
    }
}
    }
}
```

Appendix B

Samples of Simple Behavior agents

B.1 Default Behavior

```
public class DefaultBehaviorAgent extends BehaviorAgent implements DefaultBehavior {
    public DefaultBehaviorAgent() throws RemoteException {
super();
setLogLevel(LogStream.DEBUG);

makeActive();
    }

    // method needs to have call to makeActive()
    public void activate(Secret secret) {
makeActive();
    }
}
```

B.2 Vacant Behavior

```
public class VacantBehaviorAgent extends BehaviorAgent implements VacantBehavior {
    private LightManager lights;
    public VacantBehaviorAgent() throws RemoteException {
super();
setLogLevel(LogStream.DEBUG);

// devices
lights = (LightManager) reliesOn("agentland.device.light.LightManager");
// speech section
addGrammar("agentland.behavior.speechin.vacantBehavior");
addKeyNotePair("startvacant","behavior.vacant.start");
addKeyNotePair("stopvacant","behavior.vacant.stop");

// add events here
addEvent("behavior.vacant.start","activate");
addEvent("behavior.vacant.stop","actionDeactivate");
addEvent("perception.cloud.door open","actionLights");
    }

    // method needs to call makeActive()
    public void activate(Secret secret) {
makeActive();
bevSay("DEBUG","Vacant Behavior Started");
log("DEBUG", "VacantBev is now active");
    }

    // method needs to call makeListening()
    public void actionDeactivate(Secret secret) {
makeListening();
bevSay("DEBUG","Vacant Behavior Stopped");
log("DEBUG", "VacantBev is no longer active");
    }

    // add other actions here
    public void actionLights(Secret secret) {
if (secret.isA("perception.cloud.door open")) {
    try {
lights.turnOn();
log("DEBUG", "actionLights");
    } catch(Exception e) {
log("ERROR", "Error during Lights Action: " + e);
    }
}
    }
}
```

Bibliography

[1] Christine Alvarado and Randall Davis. Preserving the freedom of paper in a computer-based sketch tool. In *Proceedings of HCI International 2001*, 2001.

[2] Aaron Bobick, Stephen Intille, Jim Davis, Freedom Baird, Claudio Pinhanez, Lee Campbell, Yuri Ivanov, Arjan Schtte, and Andy Wilson. The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment. *In PRESENCE: Teleoperators and Virtual Environments*, 8(4):367–391, November 1996.

[3] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[4] B. Brumitt, B. Meyers, J. Krumm, A. Kern, , and S. Shafer. Easyliving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing*, September 2000.

[5] John Canny and Danyel Fisher. Activity-Based Computing. In *Proceedings of the Workshop on The What, Who, Where, When, Why and How of Context-Awareness, CHI 2000*, The Hague, The Netherlands, 2000.

[6] Renato Cerqueira, Christopher K. Hess, Manuel Romn, and Roy H. Campbell. Gaia: A Development Infrastructure for Active Spaces. In *Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001)*, 2001.

[7] Michael Coen. Design Principles for Intelligent Environments. In *Proceedings of AAAI'98*, pages 36–43, Madison, WI, 1998.

[8] Michael Coen. Multimodal Integration - A Biological View. In *Proceedings of IJCAI'01*, Seattla, WA, 2001.

[9] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the Computational Needs of Intelligent Environments: The Metaglue System. In Paddy Nixon, Gerard Lacey, and Simon Dobson, editors, *Managing Interactions in Smart Environments*, Dublin, Ireland, December 1999. MANSE.

[10] Trevor Darrell, David Demirdjian, Neal Checka, and Pedro Felzenswalb. Plan-view Trajectory Estimation with Dense Stereo Background Models. In *Proceedings of the International Conference on Computer Vision*, 2001.

[11] M. Dertouzos. The Oxygen Project. *Scientific American*, 282(3):52–63, August 1999.

[12] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Context-Based Infrastructure for Smart Environments. In Paddy Nixon, Gerard Lacey, and Simon Dobson, editors, *Managing Interactions in Smart Environments*, Dublin, Ireland, December 1999. MANSE.

[13] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. In *Human-Computer Interaction*, volume 16, 2001.

[14] Victor Zue et al. Jupiter: A Telephone-Based Conversational Interface for Weather Information. *IEEE Transactions on Speech and Audio Processing*, 8(1), January 2000.

[15] Jim Firby. *Adaptive Execution in Complex Dynamic Domains.* PhD Thesis, Yale University, January 1989.

[16] Jim Firby. The RAP Language Manual, March 1995. Animate Agent Project Working Note AAP-6, University of Chicago.

[17] Mark Foltz. Ligature: Gesture-Based Configuration of the E21 Intelligent Environment. In *Proceedings of Student Oxygen Workshop*, 2001.

[18] Ernest J. Friedman-Hill. Jess, the Java Expert System Shell. Technical Report SAND98-8206, Sandia National Laboratories, 1997.

[19] Krzysztof Gajos. Rascal - a Resource Manager for Multi Agent Systems in Smart spaces. In *Proceedings of CEEMAS 2001*, 2001.

[20] Krzysztof Gajos and Ajay Kulkarni. FIRE: An Information Retrieval Interface for Intelligent Environments. In *Proceedings of International Workshop on Information Presentation and Natural Multimodal Dialogue IPNMD*, Verona, Italy, December 2001.

[21] Krzysztof Gajos and Howard Shrobe. Delegation, Arbitration and High-Level Service Discovery As Key Elements of a Software Infrastructure for Pervasive Computing, 2002. In submission.

[22] Krzysztof Gajos, Luke Weisman, and Howard Shrobe. Design Principles for Resource Management Systems for Intelligent Spaces. In *Proceedings of The Second International Workshop on Self-Adaptive Software*, Budapest, Hungary, 2001. To appear.

[23] Nicholas Hanssens, Ajay Kulkarni, Rattapoom Tuchinda, and Tyler Horton. Building agent-based intelligent workspaces. In *ABA Conference Proceedings*, June 2002. To Appear.

[24] Boris Katz. From Sentence Processing to Information Access on the World Wide Web. In *AAI Spring Symposium on Natural Language Processing for the World Wide Web*, Stanford University, Stanford CA, 1997.

[25] C. D. Kidd, R. J. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner, and W. Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings*, 1999.

[26] Justin Lin, Robert Laddaga, and Hirohisa Naito. Personal Location Agent for Communicating Entities (PLACE). In *Proceedings of Mobile CHI*, 2002.

[27] Pattie Maes. How to Do the Right Thing. *Connection Science Journal*, 1(3), 1989.

[28] Marvin Minsky. *The Society of Mind.* A Touchstone Book, 1985.

[29] Louis-Philippe Morency, Ali Rahimi, Neal Checka, and Trevor Darrell. Fast Stereo-Based Head Tracking for Interactive Environment. In *Proceedings of the Int. Conference on Automatic Face and Gesture Recognition*, 2002. To appear.

[30] Alice Oh, Harold Fox, Max Van Kleek, Aaron Adler, Krzysztof Gajos, Louis-Philippe Morency, and Trevor Darrell. Evaluating Look-to-Talk: A Gaze-Aware Interface in a Collaborative Environment. In *Proceedings of CHI 2002*, Minneapolis, MN, April 2002.

[31] Alice Oh, Rattapoom Tuchinda, and Lin Wu. Meeting Manager: A Collaborative Tool in the Intelligent room. In *Proceedings of Student Oxygen Workshop*, 2001.

[32] Stephen Peters. Using Semantic Networks for Knowledge Representation in an Intelligent Environment, 2002. In submission.

[33] Brenton Phillips. Metaglue: A Programming Language for Multi-Agent Systems. M. Eng. Thesis, MIT, Cambridge, MA, 1999.

[34] Shankar R. Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Proceedings of UBICOMP 2001*, 2001.

[35] Paul Prekop and Mark Burnett. Using Activity-Centric Context to Support Ubiquitous Computing, 2002. Information Technology Division, Department of Defence, Australia. In submission.

[36] M. Roman and R.H. Campbell. Gaia: Enabling Active Spaces. In *Proceeding of ACM SIGOPS European Workshop*, Kolding, Denmark, 2000.

[37] Bill N. Schilit, Norman I. Adams, and Roy Want. Context-Aware Computing Applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[38] Bill N. Schilit and Marvin M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, September/October 1994.

[39] Steven A. N. Shafer, Barry Brumitt, and JJ Cadiz. Interaction Issues in Context-Aware Intelligent Environments. *Human-Computer Interaction*, 16, 2001.

[40] Herbert A. Simon. *The Sciences of the Artificial.* The MIT Press, 1996.

[41] Rattapoom Tuchinda. Security and Privacy in Intelligent Environments. M. Eng. Thesis, MIT, Cambridge, MA, 2002.

[42] Nimrod Warshawsky. Extending the Metaglue Multi Agent System. M. Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[43] Mark Weiser. Computer of the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[44] Brian C. Williams and P. Pandurang Nayak. Immobile Robots: AI in the New Millennium. *AI Magazine*, 17(3), 1996. Fall.

[45] Brian C. Williams and P. Pandurang Nayak. A Model-based Approach to Reactive Self-Configuring Systems, 1996. Recom Technologies, NASA Ames Research Center, MS 269-2.

[46] Terry Winograd. Towards a Human-Centered Interaction Architecture, April 1999. Working paper for Stanford project on Interactive Workspaces.