Exploring Electronic Storyboards as Interdisciplinary Design Tools for Pervasive Computing

by

Jason Forsyth

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Thomas L. Martin, Chair

Edward A. Dorsa

R. Benjamin Knapp

Elizabeth D. McNair

C. Jules White

April 30, 2015

Blacksburg, Virginia

Keywords:Interdisciplinary Design, Pervasive Computing, Electronic Storyboards

Copyright 2015, Jason B. Forsyth

Exploring Electronic Storyboards as Interdisciplinary Design Tools for Pervasive Computing

Jason B. Forsyth

(Abstract)

Pervasive computing proposes a new paradigm for human-computer interaction. By embedding computation, sensing, and networking into our daily environments, new computing systems can be developed that become helpful, supportive, and invisible elements of our lives. This tight proximity between the human and computational worlds poses challenges for the design of these systems - what disciplines should be involved in their design and what tools and processes should they follow? We address these issues by advocating for interdisciplinary design of pervasive computing systems. Based upon our experiences teaching courses in interactive architecture, product design, physical computing and through surveys of existing literature, we examine the challenges faced by interdisciplinary teams when developing pervasive computing systems. We find that teams lack accessible prototyping tools to express their design ideas across domains. To address this issue we propose a new software-based design tool called electronic storyboards.

We implement electronic storyboards by developing a domain-specific modeling language in the Eclipse Graphical Editor Framework. The key insight of electronic storyboards is to balance the tension between the ambiguity in drawn storyboards and the requirements of implementing computing systems. We implement a set of user-applied tags, perform layout analysis on the storyboard, and utilize natural language processing to extract behavioral information from the storyboard in the form of a timed automaton. This behavioral information is then transformed into design artifacts such as state charts, textual descriptions, and source code.

To evaluate the potential impact of electronic storyboards on interdisciplinary design teams we develop of user study based around "boundary objects". These objects are frequently used within computer-supported collaborative work to examine how objects mediate interactions between individuals. Teams of computing and non-computing participants were asked to storyboard pervasive computing systems and their storyboards were evaluated using a prototype electronic storyboarding tool. The study examines how teams use traditional storyboarding, tagging, tool queries, and generated artifacts to express design ideas and iterate upon their designs. From this study we develop new recommendations for future tools in architecture and fashion design based upon electronic storyboarding principles.

Overall, this study contributes to the expanding knowledge base of pervasive computing design tools. As an emerging discipline, standardized tools and platforms have yet to be developed. Electronic storyboards offer a solution to describe pervasive computing systems across application domains and in a manner accessible to multiple disciplines.

Acknowledgments

I am indebted to many people for the completion of this work: First, I owe a great deal to the faculty, staff, and students at the Virginia Tech Institute for Creativity, Arts, and Technology (ICAT). I often describe ICAT as a home for "misfit engineers and techno-phile artists". It has been a pleasure to work with Liesl Baum, Jamie Simmons, and Phyllis Newbill on the K-12 mission of ICAT. Through these experiences I have learned that weeks of work can be made worthwhile in a single student presentation. To see them succeed is a joy, no matter how difficult administrators try to stop us. To my fellow conspirators in the Sandbox, Reza Tasooji, Deba Saha, Run Yu, Michael Stewart, Chris (Orange) Frisina, Kari Zacharias, and Karl Bitkofer, it has been a joy to work, play, and watch Game of Thrones together.

I am also indebted to my first home in 3015 Torgersen Hall. To the CCMers, Tony Frangieh, David Uliana, Ryan Marlowe, Andrew Love, Krzysztof Kepa, Shaver Deyerle, and Jacob Couch, thank you for the endless games of foosball and the numerous escapes from my well-prepared lunch to instead "get happy". To my fellow e-Textiles crew, Madison Blake, Jake Dennis, Kristen Hines, Rabih Younes, Ramya Narayan, Karthick Lakshmanan, Robert Lewis, and Jacob Simmons, thank you for dealing with my lab-consuming clutter. Kristen, you have my permission to clean the lab. Just leave the monkey.

To my adopted engineering education cohort, Martina Svyantek, Homero Escobar, Lynette Wilcox, Stephanie Kusano, Cassandra Groen, Chris Ventors, and Marcia Davitt, it took several years but I might finally understand what a Cronbach alpha is. I will now avoid surveys like the plague. I'm sure grounded theory is easier, right?

Beyond my committee I had the privilege to be guided by several faculty members. The user study could not have been formulated without the help of Troy Abel. Thank you for your time and patience in shepherding a student from outside of the department and discipline. To Steve Harrison and Deborah Tatar, thank you for taking me under your HCI wing and providing advice, guidance, and support of my work. Attending DIS was a treat. To Mark Smith of the KTH, thank you for our ISWC/Ubicomp burgers and chats and for welcome assistance in my job search.

To the members of my committee: Ben Knapp, Jules White, Ed Dorsa, and Lisa McNair, this work would not be possible without your insight. Interdisciplinary work cannot flow from disciplinary advising and the perspectives each of you bring are invaluable. Ben, thank you for captaining the ICAT ship and enabling this type of work to take place.

Jules, your MDE perspective brought rigor to a process that tries to resolve ambiguity. My task would have been much harder without the framing of models and transforms. Lisa, engaging with engineering education has impacted my career. Through our research meetings I have examined my own work more systematically and with a fresh perspective I could not have received from within computer engineering. Ed, thank you for your insights into the design world and being welcoming of an interloping engineer. The initial ideas for the project came from sketches and storyboards from within Burchard. I have always enjoyed my time in Architecture+Design and will miss working with you.

To Dr. Martin / Tom, I'm not sure by which name to address you at this point. It is difficult to summarize the last seven years of work. From the product design class as an undergraduate, to the wearables course, and throughout my doctoral work, you saw some potential in me that I did not see in myself. You allowed me to wander and discover work that I was interested in. I do not know how one thanks another for such faith. I respect your work as an educator and a scholar and hope to follow your example at York.

To Katie, it's been a long journey eh? I don't think you knew what you were signing up for when this all began. All I can say is none of this would be possible without you. Regardless of a good day or bad I am happy to come home to you. I look forward to our adventures in York and the endless possibilities our future holds.

This material is based in part upon work supported by the National Science Foundation under Grant Number EEC-0935103. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Additional funding was received from the Virginia Tech Institute for Creativity, Arts, and Technology.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

"The challenge is to create a new kind of relationship of people to computers, one in which the computer would have to take the lead in becoming vastly better at getting out of the way, allowing people to just go about their lives." [2]

-Mark Weiser

The vision of pervasive computing is a world where computing technology is embedded into the clothes, objects, and environments of our daily lives. The goal of embedding this technology is to create supportive and helpful interactions with computer systems that disappear into subconscious use. However, to achieve this subtle and seamless use of computing systems, the design of pervasive computing systems must expand beyond engineering to include experts in the domains where the pervasive systems will be deployed. The purpose of this dissertation is to explore research issues in intellectual tools for supporting interdisciplinary design of pervasive computing systems.

Through our work teaching courses in interactive product design [3], physical computing, interactive architecture, and through the work of others in interdisciplinary design, we have found that rapid prototyping and generating large numbers of ideas is key to exploring a complex design space and finding good solutions. Prototyping serves two important purposes: to gain knowledge of the material and processes used in a domain [4], and to help identify good solutions in a highly constrained domain [5, p.650]. The speed in which prototypes are created is critical to the design process as multiple iterations of prototyping often outperform single iterations [6].

This dissertation examines the current state of pervasive computing prototyping tools and finds that current tools fall short for several reasons. First, few tools support a "shared view" of the design space that will allow users from different backgrounds to concurrently express design ideas. Second, for those tools that do support a "shared view", they are either domain specific tools or simplified programming languages. Even a simplified programming language is a barrier for team members who are not programmers as their understanding of the behavior of the prototype is solely dependent upon how well they understand the language. Furthermore, domain specific tools cannot handle the various types of prototypes that could be expected by a design team. For example, tools intended for location-based systems may fare poorly in creating wearable applications.

This dissertation explores the research issues involved in developing a novel tool that will fit within the interdisciplinary design paradigm for pervasive computing products. This tool will not generate final products, but is instead intended to generate prototypes and design artifacts that enable the rapid evaluation of design ideas during the product's design and development. Specifically we propose that storyboards, and storyboarding in general, be used as a design and prototyping medium for interdisciplinary teams. Storyboards convey context, location, action, and temporal phenomenon [6] [1, p.296] that cannot be expressed by current programming tools. These temporal and contextual phenomenon are key aspects of the user experience for pervasive systems. Storyboards have been shown to be an effective communication tool in an interdisciplinary setting [7, 8] and contain sufficient formal structure to be the basis for several existing programming tools [9, 10, 11, 12]. Furthermore, storyboards incorporate drawing into the design process. Drawing is a fundamental aspect of design [1, p.95], and through drawing, designers can envision and work out problems [13].

1.2 Research Questions and Contributions

We examined electronic storyboards through our own teaching experiences, surveys of existing literature, and a usability study. This section outlines specific research questions addressed in this dissertation and the subsequent contributions.

1.2.1 Research Questions

RQ 1: How are the needs of interdisciplinary teams supported by current pervasive computing design tools?

Chapter 2 outlines challenges faced by interdisciplinary teams when developing pervasive computing systems and identifies difficulties in design materials, tool, and processes. Recognizing the importance of prototyping to the success of interdisciplinary teams, we conduct a survey of existing pervasive computing design tools and find that many fail to meet the requirements for interdisciplinary teams. We propose electronic storyboards as a solution to these problems and explore their implementation in the next research question.

RQ 2: How can information in storyboards be extracted to create design artifacts?

Chapter 3 outlines a model-driven approach to developing electronic storyboards. A domain-specific modeling language is developed in Eclipse [14] to explore how information in storyboard layout and content can be extracted to produce design artifacts such as source code, state charts, and textual descriptions. A multi-device model of pervasive computing systems is presented to capture a range of pervasive interactions. A timed automaton model of computation is used to capture storyboard information and derive design artifacts.

RQ 3: What is the impact of electronic storyboarding on interdisciplinary teams during prototyping?

Chapter 4 examines team collaboration with electronic storyboards using boundary objects. These objects have been studied extensively in computer-supported collaborative work (CSCW) and are extended to pervasive computing design to examine the efficacy of electronic storyboarding for interdisciplinary teams. Chapter 5 examines the results of the study and shows how electronic storyboards support design expression and reflection for interdisciplinary teams. Additionally, we evaluate the models described in Chapter 3 and provide guidelines for future design tools in Chapter 6.

1.2.2 Contributions

Overall this dissertation makes two contributions. First, it contributes to the discussion regarding the forms and affordances of pervasive computing design tools by advocating storyboarding as an accessible and interdisciplinary design tool. Following the guidelines from Dow et al. [6] we show how electronic storyboards support the "multiple views" requirement for pervasive design tools. Furthermore, our research attempts to answer Gregory Abowd in his critique of pervasive computing where he asks, "What is [pervasive computing's] Hypercard?" [15], in reference to the successful programming language for desktop environments. This discussion is important to pervasive computing as the field matures thereby making the vast "invisible computing" vision of Weiser [16] ever more tangible. However the field currently focuses more heavily on mobile computing to the exclusion of further hardware development [17]. The dissertation advocates for more development of novel pervasive systems to more fully explore the potential design space.

Second, this dissertation provides methods for iteratively refining ambiguous specifications while keeping human expertise in the loop. As electronic storyboards are "compiled" the user is repeatedly queried as to which information is important to defining the device's behavior. This approach can be extended to additional areas where model transformations are "misaligned" between different sets of knowledge. Different meta-models may be required but our approach of domain-specific annotations coupled with natural language processing and user queries can be applied to partial models [18] and situations where other software specifications must be refined.

To aid these arguments we make several technical contributions:

1. *An information extraction method for deriving timed automata from storyboards.* In Chapter 3 we define a domain-specific modeling language for electronic storyboards and implement model transformations to derive timed automata from those storyboards. Information extraction and transformation presents new methods to handle ambiguity in storyboards and iteratively queries the user to resolve missing or ambiguous information.

2. *An analytical model of pervasive computing prototyping using boundary objects to examine the efficacy of electronic storyboards for interdisciplinary teams.* In Chapter 4 we present a model of pervasive computing based upon three properties: computation, user interaction, and physicality. We examine how those properties are expressed by teams when using electronic storyboards as boundary objects. By examining what information teams "push" into the boundary object, e.g. what they draw and tag during the user study, and what discussions take place during the study, we can assess the impact of electronic storyboards on the study tasks.

3. *Best practices and development guidelines for future interdisciplinary tools.* Based upon the results of the user study in Chapter 5 we provide guidelines for future tools similar to electronic storyboards that infer design artifacts from informal sketches and storyboards. We advocate for the importance of free-form sketching in future tools, further isolation of the end-user from underlying models of computation, and further refinement of UI elements rather than tool outputs.

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows: Chapter 2 outlines the challenges faced by interdisciplinary teams while developing pervasive computing systems. We identify the need for prototyping tools that help establish a shared language and can work in a general setting. Addressing these issues we propose electronic storyboards. Chapter 3 describes our model-driven approach to developing pervasive computing systems. We present a model based upon timed automata and provide an information extraction method to derive automata from storyboards. Chapter 4 models electronic storyboards as boundary objects to evaluate their impact on interdisciplinary teams. Using these shared artifacts we develop a user study to determine how electronic storyboards support design discussions and iterations. Chapter 5 presents the results of a user study with six interdisciplinary teams and analyzes support for design discussions and iteration, assesses general usability, and determines the efficacy of our underlying models. Finally, we conclude in Chapter 6 with a summary of this work and look towards future research.

Chapter 2

Background and Related Work

This chapter addresses Research Question 1 by examining the needs of interdisciplinary teams and the support offered by existing design tools. More specifically, Section 2.1 describes the origins of pervasive computing and advocates for interdisciplinary design of pervasive systems. Section 2.2 identifies specific challenges faced by interdisciplinary teams due to unfamiliar design materials, differing design processes, and motivates the importance of prototyping. Section 2.3 examines the current state of pervasive design tools. Finally, Section 2.4 summarizes our findings and proposes electronic storyboards that will be discussed in the next chapter[1].

2.1 Need for Interdisciplinary Design

Pervasive computing has been a driving vision for computing research for nearly 20 years. Pervasive computing was first described by Mark Weiser in which he put forward a world fully embedded with sensing, computation, and networking [20]. What is revolutionary about Weiser's vision is that embedding computing into our clothes, environments, and daily lives, can be calming. Since that time technological advances in inexpensive wireless networks and embedded computing have provided a platform on which to build pervasive computing [21].

Pervasive computing can be viewed as the convergence of distributed systems and wireless computing. Technologically it retains the challenges of those disciplines, but inherits several new challenges such as smart spaces, invisibility, localized scalability, and masking uneven conditions [21]. Of these research challenges, what remains elusive is the calming, or "invisible" use of computing. Historically as technologies become adopted and more refined they move into the periphery of our attention [16]. The use of the technologies becomes subconscious and is only noticed if something is absent or amiss. Examples of technologies that have moved into periphery include electricity, motors, and writing. Each of these technologies can be used effortlessly and are only brought to our attention when they

[1]Portions of this chapter are based upon the author's work in [19] and are reproduced here under copyright permitted by Emerald Publishing.

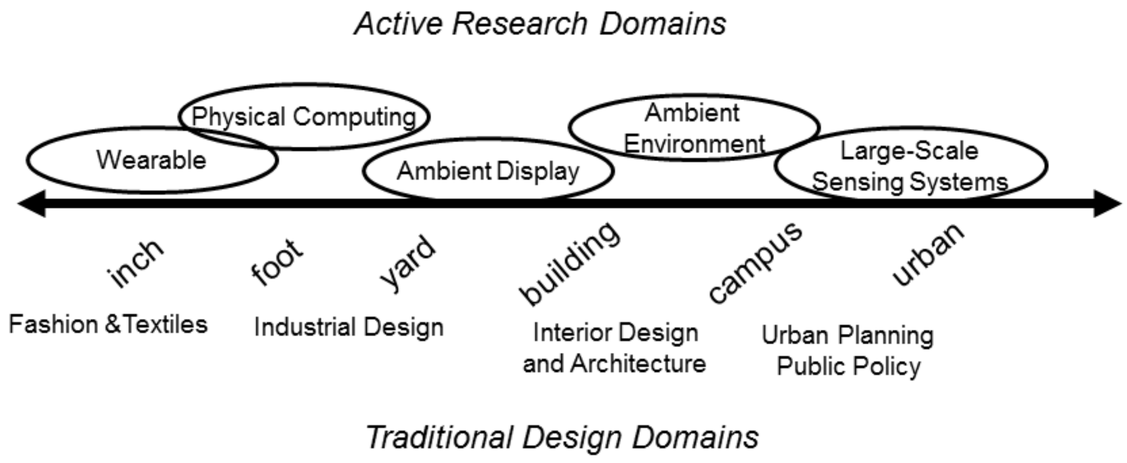


Figure 2.1: Taxonomy of Pervasive Computing Domains

are absent or do not work [20]. The goal of pervasive computing is to make interactions with computing technology as subconscious, or invisible, as these current technologies. Weiser said that our computers should be "an invisible foundation that is quickly forgotten but always with us, and effortlessly used throughout our lives." [22].

Creating these "invisible" interactions requires detailed knowledge about the needs and desires of the end user. To understand these needs, designers of pervasive systems must "uncover the very practices through which people live and to make these invisible practices visible." [23]. We believe that achieving invisible interactions requires an interdisciplinary approach that includes not only researchers and engineers, those who develop pervasive technologies, but also practitioners in the humanities, arts, and design. It is this later group who are trained in understanding human behavior and the subconscious motivations and interactions of the end user [2, 22]. Working in an interdisciplinary setting reflects the foundations and goals of pervasive computing by combing a revolution in technology with a focus on human experience that makes human-computer interaction calming and supportive.

To properly implement pervasive systems they must be designed to address human needs and concerns. It is not enough to possess the proper technology to create these systems, but that technology is employed in a well designed manner that creates enjoyable, useful, and appropriate interactions. Design requires an understanding and balancing of relevant constraints to find a proper solution [5]. In addition, the vast scale of pervasive computing means this design will happen across multiple domains including fashion, industrial design, architecture, and urban planning. While computer scientists, engineers, and technologists may possess the knowledge to implement pervasive systems, they may not possess the domain-specific knowledge at each of these scales of interaction required to bring human concerns to the forefront of design. These technical experts must collaborate with domain-experts within these scales to address both technological and human constraints that are required for successful design. In short, the design of pervasive systems must be interdisciplinary.

While interdisciplinary design is not a widely recognized requirement for pervasive computing, it is a logical extension of the field. Because pervasive computing deals intimately with the physical and human world, the existing areas of pervasive research tend to align with traditional design domains. If we consider Weiser's scales of inch, foot, and yard [20], these "computing scales" roughly align with active research areas regarding wearable, tangible, and ambient interactions. Continuing our analogy, those three disciplines and interaction scales, also align with the practices of fashion design, industrial design, and architecture. Combining these elements on a single axis and extending it to include large scale cyber-physical systems provides a taxonomy of pervasive computing shown in Figure 2.1. This taxonomy provides natural associations between engineering and design disciplines. This close association is most frequently displayed in the wearable computing community in which fashion and textiles professionals are a common presence. Having outlined the need for interdisciplinary design, we discuss some of the challenges faced by by interdisciplinary teams in the next section.

2.2 Challenges in Interdisciplinary Design of Ubicomp

In this section, we outline three areas of difficultly for interdisciplinary design. First, we discuss the need for all parties to have an understanding of the "materials" used. Second, we discuss disciplinary differences in problem solving that can cause difficulty between engineering and design. Finally, we highlight the importance of prototypes and prototyping tools for achieving better design outcomes.

2.2.1 Design Materials

"The technical terms used by engineers are not the normal language of the industrial designers - indeed they may find them meaningless. Industrial designers, on the other hand, express their ideas and describe materials in ways that, to the engineer, sometimes seem bewilderingly vague and qualitative" [24].

-Materials and Design, 3rd Edition

Design is a conversation with materials [25]. Through sketches, diagrams, models, and prototypes, designers concurrently "see" and evaluate design solutions [13]. This conversation is "reflective" and the materials "talk back" with new insights and realizations [26, 27]. For pervasive computing systems, this conversation involves materials that may be foreign and unknown to designers. Given the embedding of technology into daily lives, one may believe they understand it, however many hold incorrect "folk theories" about technology based upon marketing, television, or social connections [28]. It is equally likely that engineers or scientists would have difficulty working with wood,

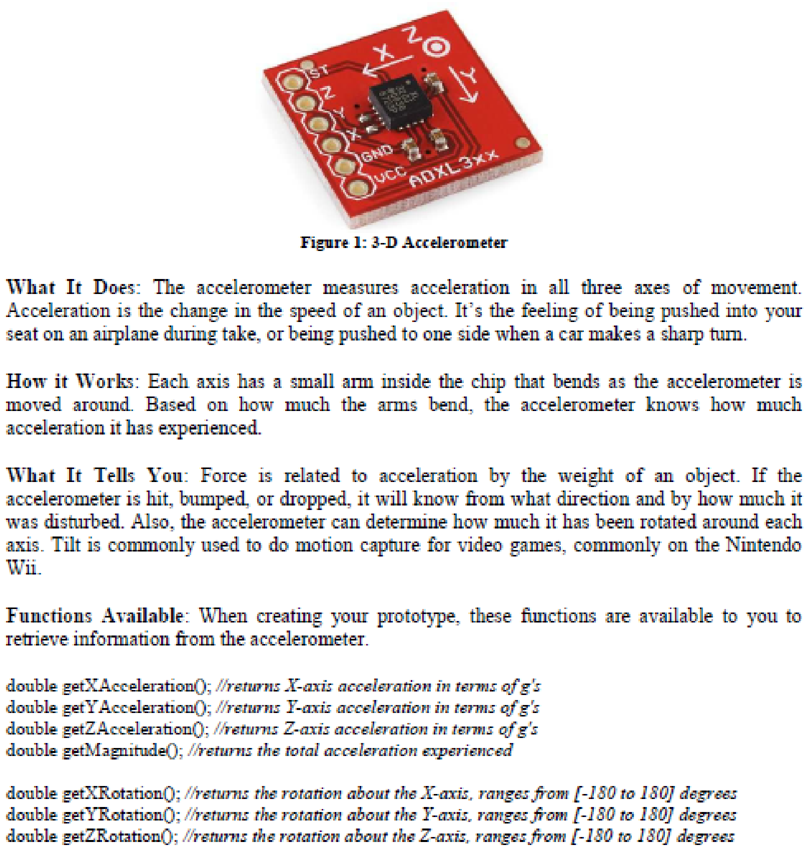


Figure 1: 3-D Accelerometer

What It Does: The accelerometer measures acceleration in all three axes of movement. Acceleration is the change in the speed of an object. It's the feeling of being pushed into your seat on an airplane during take, or being pushed to one side when a car makes a sharp turn.

How it Works: Each axis has a small arm inside the chip that bends as the accelerometer is moved around. Based on how much the arms bend, the accelerometer knows how much acceleration it has experienced.

What It Tells You: Force is related to acceleration by the weight of an object. If the accelerometer is hit, bumped, or dropped, it will know from what direction and by how much it was disturbed. Also, the accelerometer can determine how much it has been rotated around each axis. Tilt is commonly used to do motion capture for video games, commonly on the Nintendo Wii.

Functions Available: When creating your prototype, these functions are available to you to retrieve information from the accelerometer.

double getXAcceleration(); *//returns X-axis acceleration in terms of g's*
double getYAcceleration(); *//returns Y-axis acceleration in terms of g's*
double getZAcceleration(); *//returns Z-axis acceleration in terms of g's*
double getMagnitude(); *//returns the total acceleration experienced*

double getXRotation(); *//returns the rotation about the X-axis, ranges from [-180 to 180] degrees*
double getYRotation(); *//returns the rotation about the Y-axis, ranges from [-180 to 180] degrees*
double getZRotation(); *//returns the rotation about the Z-axis, ranges from [-180 to 180] degrees*

Figure 2.2: User-Friendly Datasheets

metals, or plastics of traditional design materials. Furthermore, scientists and engineers are not trained to understand the human-dimensions of design.

Because the computational, physical, and user experience of a pervasive system are tightly bound, all participants on a design team must have a "shared view" of the design space, that is, computing and non-computing team members must have a sufficient understanding of how their design choices impact the performance of the overall system. Having a shared view of the design space is important in prototyping as each discipline has concerns that must be expressed. While non-computing participants do not require the same level of technical understanding as engineering team members, they still require a certain level of abstraction to understand why the prototype behaves the way it does. Having the proper level of abstraction about the operation of the system allows team members to reason about why a behavior did not work. Additionally, when prototyping the systems, the non-technical members can be involved in the creative process instead of simply instructing the engineer about the desired behaviors.

We personally encountered this problem while teaching an interactive product design class for senior-level marketing, industrial design, and computer engineering students [3, 29]. One solution to establishing a "shared view" was Arduino-based prototyping exercises, led by computing engineering students but aimed at the marketing and industrial design students. Each team was provided an initial "dull" prototype and was challenged to create a children's toy. Teams were provided “user-friendly” datasheets to help explain the abilities offered by the prototype’s sensors. A sample datasheet is shown in Figure 2.2. This exercise helped non-computing participants develop a sense of the potential of pervasive computing systems.

2.2.2 Design Processes

“Like the sirens who tried to lure Ulysses to destruction, these myths lead us to the false assumption that we can adopt a process that will take us along a straight path from intention to implementation [1].”

-Bill Buxton

Another challenge working in interdisciplinary design is nature of design problems and the different problem solving processes employed by science, engineering, and design disciplines. Design problems are “wicked problems” that are often poorly specified or ill-defined [30]. As examples, design problems such as “juvenile diabetes”, “workplace wellness”, and “home energy efficiency” admit many solutions, constraints, and evaluation criteria. Design problems are in contrast to traditional science and engineering problems which postulate a set of rules or hypothesis to generate a solution. Often based upon the constraints selected, there is one “correct” solution, whereas in design there might be a variety of acceptable solutions. Schon describes these two problem types as “problem setting” versus “problem solving” [26]. In product design, Buxton contrasts the two as “getting the right design” and “getting the design right” [1, p.78]. In addressing these two problems, design and engineering employ different problem solving methods. Summarizing Cross [30, p.19] and Lawson [31], *design thinking* is “abductive” or “appositional” and adopts solution-focused methods to generate and test ideas whereas *scientific thinking* is “deductive” and “inductive” and adopts a problem-focused approach. One does not naturally adopt a particular mode of thinking, but the approach is learned through training and education.

In a product design setting such as our interdisciplinary product design class [3], both “problem setting” and “problem solving” is required. Teams must identify a compelling product to create but also determine one that is feasible to implement. While each problem will have different signficance during the process, there is no explicit transition or handover between “solving” and “setting”; likewise there will always be insight and engagement with design and engineering. Buxton [1] provides a graphical representation of how these roles change during product design in Figure 2.3. As design and engineering are always active elements of the process, the disciplinary framing and problem solving methods these disciplines may come into conflict. Given engineer’s high uncertainty avoidance [3], they may quickly seek solutions to problems before all alternatives are tried [4]. Adopting a solution too soon can be damaging for the design process as the more prototypes developed over time, in general the better the design outcome [32].

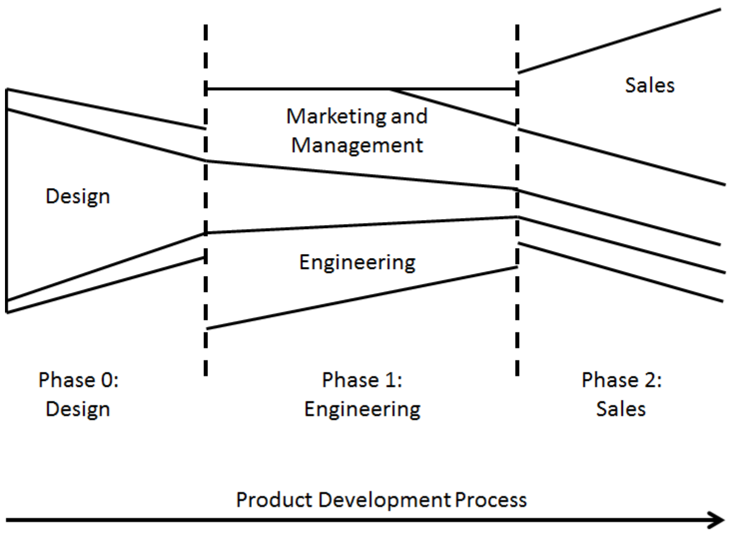


Figure 2.3: Buxton's roles of engineering, design, and marketing during product design [1].

We personally observed difficulties between these two problem solving methods during our product design class. During the ideation phase, engineering students would frequently latch on to a potential solution or criticize new ideas prematurely. To address this problem, we prohibited engineers from saying "no" to idea for technical reasons or deciding on a solution for several weeks. This restraint helped protect new ideas and allowed a variety of potential solutions to move towards prototyping. We discuss the importance of prototyping in more detail in the next section.

2.2.3 Design Tools and Prototyping

"I believe that if we think about people and then try, try, and try again to prototype our designs, we stand a good chance of creating innovative solutions that people will value and enjoy." [5, p.643]

-Bill Moggridge

From Buxton's diagram in Figure 2.3, the role of engineering and marketing become more prominent towards the end of the design phase. During this period, teams are realizing their ideas into prototypes to evaluate potential design ideas. Prototypes take many forms that range from sketches about a product concept, to foam models examining form, and interactive prototypes to experiment with different behaviors. Regardless of their differences, each prototype helps to realize and examine an important aspect of an idea. Producing large number of prototypes is critical to achieving positive design outcomes. In creating prototypes, the design team becomes more familiar with the materials and processes of the design domain [4] and helps identify solutions in a highly constrained environment [5, p.650]. This process of repeatedly producing, examining, and re-evaluating prototypes continues until intuition takes over and people begin to engage with the prototype, rather than critique [5, p.726].

Buchenua and Suri identify three different kinds of prototypes: “looks like”, “behaves like”, and “works like” [33]. With pervasive computing products being a close coupling of user experience and technology, prototyping “behaves like” and “works like” aspects of a product are important. While tools have been developed for interdisciplinary teams to prototype interactive behaviors, often these tools require all members of the team to learn a programming language or only work within a narrow application domain. The lack of appropriate tools affects the design cycle, and the product itself, by increasing the time between prototypes and limiting the number of prototypes when under time constraints. Given the importance of prototypes, and consequently prototyping tools, we evaluate their current status in the next section.

2.3 Survey of Pervasive Computing Design Tools

Having established the motivation for interdisciplinary design of pervasive computing, we now present a broad survey of existing pervasive computing tools. The metrics for these tools are largely informed by the work of Dow et al., who conducted a survey of practicing designers to determine desired tools’ properties [6]. We have condensed their seven categories into five to better represent tool properties of interest and have expanded the understanding of several categories by creating subcategories. In general these guidelines cover the topics of (1) how to design an interactive or pervasive application, (2) how events are defined within that application, (3) how to understand the abilities of sensors and which ones are available, (4) how these tools support different hardware platforms or simulations, and (5) what existing design practices are supported within these tools. These five categories are shown in Table 2.1 with the listing of each tool and which properties are supported. In Section 2.3.1, we describe the tool properties to be evaluated. In Sections 2.3.2-2.3.6, we describe how existing tools meet these properties.

2.3.1 Tool Properties Under Evaluation

Multiple Representations: A key difficulty in interdisciplinary design is the “language” or representation used to express a design. Often interactive products are created using a textual programming language, which is familiar to the engineers, but unfamiliar to the designers. Having multiple, and possibly simultaneous, representations of a design overcomes this problems by allowing all members of a design team to participate in the design of an application and have a view that is suited for them. Additionally, multiple views can combine the benefits of different programming representations. For example, visual programming and data flow metaphors are better suited to describing continuous actions, while traditional sequential programming is appropriate for conditionals. We will review three instances of this property including tools that allow simultaneous visual/textual, state charts/textual representations, and tangible representations.

User Defined Events: Pervasive applications depend heavily on sensor data gathered from the physical world. These physical events can be as simple as the state of a switch, or they can be more complex events such as the classification of an activity. Regardless of the event in question, fusing sensor data into meaningful information is a difficult task. For simple events, thresholds can be set on sensor data, but more complex events must be analyzed with machine learning algorithms. Several tools have developed methods to aid in synthesizing events from complex sensor data. We examine tools that allow events of interest to be demonstrated by the user, expressed through natural language, constructed using signal processing data flows, and delivered to the application through the use of a context infrastructure.

Knowledge Support: Combining sensing and computing introduces a new material into the design process that is unfamiliar to non-computing experts (Sec. 2.2.2). Dow et al. motivates this category by discussing how professional designers require information about current and upcoming technologies to inform their design. Additionally, when performing collaborative design, all parties must understand how the underlying technology behaves. Under this guideline we examine two sets of tools that allow non-engineering practitioners to learn about new technologies through play and hands-on experience, as well as customized data sheets tailored for non-engineers.

Testing and Device Support: Depending on the scale of a pervasive system, it may not be possible to have all devices or services present during design. Still, the system will need to be tested using real or generated events. This supports the creation of light-weight prototypes that are not fully formed, and more finalized designs that must be simulated. Within this guideline we examine pervasive tools that support device emulation, simulation of pervasive spaces, and provide Wizard-of-Oz methods to generate input for a design.

Integration with Current Practice: Finally, we examine tools that integrate current design practice into creating pervasive applications. This approach keeps designers within the normal realm of design and lowers the barriers to implementing interactive systems. Here, we examine practices that leverage existing design practice through foam and paper prototyping, sketching, storyboards, and textile fabrication. Additionally we look at existing computing practices that have been recently adopted into design, namely Processing [34] and the Arduino family of products [35].

2.3.2 Multiple Representations

As described in Section 2.2.1 and by Dow et al., a desirable property of a design tool is to provide a shared view of the same design. This shared view can take several forms: either there are multiple design languages that describe the same design, or there are multiple lenses from which the design can be viewed. The first approach, multiple languages, is generally achieved by having programming languages that allow increasing levels of abstraction that hide lower-level implementation details. A shared view has a single underlying program but can be viewed or expressed in different

Table 2.1: Evaluating Interdisciplinary Tools for Pervasive Computing

Tool Name	**Multiple Representations**	**Event Description**	**Knowledge Support**	**Device Support**	**Current Practice**
dTools [36]	●	●		●	
Sketchify [37]	●				●
Activity Designer [38]		●		●	●
Topiary [39]		●		●	
Papier-Mache [40]		●		●	
Lingua Franca [41]	●				
ESPrantoSDK [42]	●				
iCATch [43]	●				
TileScript [44]	●				
Modkit [45]	●				
aCAPpella [46]		●			
Bosu [47]		●	●		
CAMP [48]		●			
Ubidesigner [49]		●			
iCAP [50]		●			
GART [51]		●			
CRN Toolbox [52]		●			
VisualRDK [53]		●			
Phidgets [54]		●			
xTel [55]		●			
iStuff [56]		●			
PTK [57]		●			
Inspirational Bits [58]			●		
User-Friendly Datasheets [29]			●		
Ubiwise [59]				●	
DiaSim [60]				●	
Tatus [61]				●	
Calder Toolkit [62]					●
Arduino [35]					●
Processing [34]					●
Lilypad Arduino [63]					●
Ex-A-Sketch [64]					●

ways. Each feature is useful in interdisciplinary design. In this section we describe three different manifestations of this property. The sub-categories we consider are: visual/textual, state chart/code, and tangible programs.

Visual & Textual Representations

These tools generally provide a set of blocks that can be connected together to write a program. This visual program is directly linked to or produces a textual representation of the program. This shared view allows novice programmers to code with the visual blocks, but also allows a more experienced programmer to write code in the same environment. Tools that exhibit this property are Lingua Franca [41], EsprantoSDK [42], iCATch [43], and ModKit [45][65].

Generally high-level functions are encapsulated into visual blocks that are connected either through a puzzle-piece metaphor, such as with EsprantoSDK and Modkit, or linked together in a dataflow manner such as LinguaFranca and iCATch. The puzzle-piece can be constructed to ensure only related blocks can fit together thereby reflecting syntax restrictions that are present in the underlaying language. For example, a piece that reads a value from a sensor might be shaped such that it could only be connected to a variable block in which to store the sensor reading. Dataflow languages do not necessarily have this ability, but syntax can be enforced with proper type checking when values are passed between blocks.

In addition to these two forms, EsprantoSDK provides a middle "macro" layer that exists between its visual and textual representation. This marco layer is intended as a middle step for domain experts who know the functions they want to use but do not want to program in a full textual language. For example, the macro *drawTiles* will take a list of coordinates and display tiles on the screen. Using this macro provides more flexibility than the visual language but is not as complex as the underlying textual language.

States & Code

Sketchify [37] and dTools [36] provide two representations of a program in the form of state charts and traditional textual programming languages. dTools allows the designer to describe the behavior of a prototype using statecharts where each state is a visual representation of the prototype at that point in time. Items such as buttons, sliders, and LEDs can be visually represented on the sketched layout of the prototype. Similarly, Sketchify allows designers to sketch graphical interfaces and define the transitions between these interface views. For each tool, the visual representation describes a particular state of the prototype at a given point in time, and combined with the transition sequences, is approximate to a state machine. Underlying each visual representation is textual language that offers more explicit control. Within dTools each state can be extended by using Java APIs that provide access to key events such as entering or exiting a state, or when a sensor value is updated. Similarly, states in Sketchify can be replaced with scripting "spreadsheets" to allow greater control of state behavior.

Tangible

An interesting representation for interdisciplinary teams are tangible programs where the program logic and behavior has a physical representation. One example, Tangible Programming Bricks [66], allows users to stack LEGO bricks to form a program. Each brick represents a single function, such as count, add, or display, that can be executed by the program. To allow customization of the program, additional cards could be inserted to allow variables or parameters. A similar approach using blocks that link together has been used to make a physical representation of the Lego Mindstorm Quetzal language [67]. A particular advantage of the approach in [67] is the use of coiled wires to link separated blocks of code allowing for the introduction of conditional statements. Finally, Boda Blocks [68] allows exploration of cellular automata by combining a visual rules interface with stackable color changing blocks. The color of the block indicates its current state and allows the user to view immediate feedback about their created rules.

Discussion

A particular issue when using multiple representation is the amount of linkage or coupling between the two representations. Given two representations, is there a direct transformation from one to the other, or does one representation hide details from the other? For example, all three expressive layers in EsprantoSDK are directly linked such that changes in one layer (visual, macro, or textual) are directly reflected in the other. Alternatively, states within dTools can be augmented by Java code, but this argumentation is not reflected in the visual layer because the visual form is of the physical prototype and not its behavior.

Neither approach is inherently better than the other but the issue of linkage must be taken into account when designing multiple representations. Providing a direct link between all programming layers could become cumbersome if low-level details or other elements that don't directly affect the prototype's behavior must be shown. However, propagation of information between layers is key to maintaining a shared view. The question of what information should be propagated is largely based upon the purpose of each layer. Low levels details, such as a selected processor or sensor, need not be expressed in higher layers unless that choice impacts the behavior of the prototype. Selecting a slower processor might be necessary to reduce power consumption, but that change should be reflected in additional layers if performance is impacted.

2.3.3 Event Description

Pervasive systems are implicitly and explicitly interactive. As such, a majority of their behavior is predicated on gathering and processing sensor data. Depending on the complexity of the physical event, creating meaningful events from the raw sensor data might be a difficult task. For simple applications the event may be recognized by a sensor value passing a threshold, as in monitoring temperature. However, more complex events, such as activity or context recognition, need detailed signal processing that may require expert knowledge. In this section, we overview four different approaches to the definition of events. We overview methods in demonstration, natural language, data flow programming, and APIs.

Demonstration

Defining events by demonstration requires that the user, or set of users, perform the activity and passes that information to a training algorithm for classification. For demonstrated events, the user must segment the data and indicate to the system the beginning and end of the desired activity. This is accomplished in dTools [36] through a foot pedal, which when depressed, indicates the activity is occurring. Alternatively, a CAPpella [46] requires that the user review streams of data that are aligned with already recognized activities that have been observed in a room of interest. The user then indicates which input streams are relevant to the recognized activities and which activities should be performed. Once the activities have been annotated, the tools use machine learning algorithms to learn the appropriate responses. dTools uses Dynamic Time Warping and a CAPpella uses Dynamic Bayesian Networks.

For dTools and a CAPpella, learning by demonstration is a feature of a larger programming platform; however, for certain programs, the demonstrated activity is the program itself. This is the case for Bosu [47] which is a toolkit for physically responsive applications. Bosu modules are different shapes embedded with a shape memory alloy that can record and playback interaction that has been demonstrated on them. A user can take a module, perform the desired behavior, and playback the behavior by pushing a button on the module or using a remote control.

The benefits of the demonstration approach is that users can quickly define arbitrary events and focus on the overall behavior of the prototype. Having only a limited set of events might be restrictive for some designers [69]. Underlying these Demonstration techniques is a machine learning algorithm that performs the recognition. If the event of interest is not suitable for that algorithm, then this approach will fall short. Furthermore, machine learning algorithms insert ambiguity into the event description as each algorithm is subject to recognition errors. To avoid these situations some indication must be provided to the application designer that reveals the inner state of the algorithm, such that they can better train the recognizer. Ongoing research in interactive machine learning is seeking to address these issues [70, 71, 72, 73].

Natural Language

Natural language approaches to event description construct sentences that describe events of interest to the application designers. These tools create rule-based systems that operate on a fixed vocabulary or rule structure to derive the intended behavior. Tools that exhibit these properties are CAMP [48], Activity Designer [38], Ubidesigner [49], Topiary [39], and iCAP [50].

Generally the structure of the rules created are of the form “If <event> then <action>”, where the events and actions are selected from a set of pre-determined actions. Each tool provides a rule-editor that allows the events and actions fields to be specified with system defined events.

For each of these tools, the ease of constructing natural sentences comes at the expense of having a limited “vocabulary”. Within CAMP there is a limited set of interrogative words based around “Who, What, Where, When, Why” objects or events that the system understands. New words can be created but they are based on existing words provided by the system. Additionally, most rules are resolved to Boolean values such that the statement is true or false.

For many events or applications this Boolean resolution cannot be achieved because the system can only have a probabilistic understanding of whether an event has occurred. Activity Designer attempts to resolve this problem by allowing statements to include ambiguity in their construction. This “certainly threshold” can take the form of an event being “likely”, “unlikely”, “very likely”, which helps retain the ambiguity present in the system and provides the designer and user alike with better insights into the underlying nature of the program.

Signal Processing Blocks

Another approach to defining events is to allow the user to conduct their own data collection and signal processing but provide modular blocks to perform elements of activity recognition, such as filtering, signal segmentation, and training. GART [51] provides Java API calls that allows access to sensors, machine learning algorithms, and libraries to store recorded data. Similarly, the CRN Toolbox [52] provides a set of data flow blocks that provide similar abilities to GART, but are based in a visual programing language.

While blocks that segment data streams, train classifiers, and run activity recognition are very useful, they are only useful to those who are already familiar with the theory and practice of signal processing. These approaches are useful for practitioners in the field, but they may not be ready for interdisciplinary design teams that will require instruction and guidelines on how to create recognition applications.

Infrastructure/API

A different approach to dealing with events is to remove the user entirely from their definition. Many tools treat context and activities as events that are defined by another entity and are then provided to the user. Receiving, or being notified of an event, takes the form of asking an operating system or some other “context server” to notify or broadcast when an event occurs. The acquisition, reasoning, and delivery of context is performed elsewhere, but from the perspective of the application designer, context is a response to an event or API call. The tools that follow this paradigm are Context Toolkit [74], VisualRDK [53], Papier-Mache [40], Phidgets [54], iStuff [56], PTK [57], and the popular Android platform [75].

Some tools allow more access and control to the delivery of context than others. The Context Toolkit is a framework for determining and delivering context, but context is viewed as an event from the application designer's perspective. Similarly, iStuff has an event switch board that ties together input and output events. Other tools, such as VisualRDK, do not attempt to resolve context, they simply rely on another service to deliver the events. These approaches provide different levels of user involvement depending on their needs and experience in defining and creating context.

Overall the separation of concerns between generating and delivering context allows for more modular and standardized context applications. Delivering context as events or callbacks within a tradition programming language (such as within Java in the Context Toolkit) lowers the barrier of entry to developing context applications. Users who are already familiar with the language can now build context-aware applications. However, the drawbacks begin when the delivered context is not the desired information or is not suited for a particular application. In this case, the context recognition algorithms will need to be re-evaluated, which may present significant difficulty.

2.3.4 Knowledge Support

To effectively design in a domain one must have hands-on knowledge of the materials and processes (tools) of that domain [4]. While many of the tools we have surveyed provide means to construct pervasive applications, many potential users are unfamiliar working with computing and sensing elements. To overcome these problem, first the design teams must understand the basic properties of the computing and sensing elements they are using. Furthermore people often hold incorrect assumptions about the abilities of certain technologies, based on marketing, television or social connections [28]. Here we review two approaches that endeavor to teach non-experts about computing and sensing elements.

On way to gain knowledge about the material is though hands on prototyping. Martin et al. conducted a short prototyping exercise with interdisciplinary teams of undergraduate students convey the abilities offered by computing and sensing elements [29]. Unique to their approach is that each prototyping team was given a "user-friendly datasheet" that explained the abilities of each sensor in terms that all team members could understand. Each data sheet described the sensors or actuators in terms of "What It Does", "How It Works", "What It Tells You", and listed the available functions to control or receive information from the sensor. An example of these datasheets is shown in Figure 2.2.

A different approach to hands-on prototyping is to provide ready-made examples about certain technologies. Inspirational Bits [58] exposes potential users to the properties of technologies through demonstrations. One particular demonstration exposed the range and sensitivity properties of RFID tags through a dice game where one side of the dice was embedded with a RFID tag.

2.3.5 Device Support

Design tools must support practices that allow debugging and testing of pervasive applications. In this section we overview pervasive simulators and emulators to test environments, devices, and Wizard-of-Oz (WoZ) techniques to enable debugging of applications when real sensor data is not available.

Emulation & Simulation

Emulators provide a virtual platform in which to demonstrate and debug an application. Both Activity Designer [38] and Adobe Device Central [76] provide emulators to test applications on mobile phones. Activity Designer provides a front view of the mobile phone where the user can interact with the buttons on the phone. Additionally, noise can be added to generated sensor inputs to add more realism to the test. For professional mobile application development, both Apple and Android provide full emulators of their mobile platforms. Similar to Adobe Device Central, the Apple iOS and Android emulators provide fully interactive forward-views which are controlled by the user as if it were a real phone. Additionally, users can simulate changes in locations and sensor values.

While emulation provides testing of a single interface or device, simulation allows for a richer experience that can test features such as user interaction in a 3D space [59] [77], validation of context rules [78], and network performance estimation [61] [79] [80]. Ubiwise [59], one of the first pervasive simulators, allows users to navigate a 3D world and interact with virtual objects. This interaction assumes a device is mediating this interaction and allows the prototyping of interfaces within the expected environment. Similarly, a hybrid prototyping approach has been used to combine virtual environments with a physical miniature thereby removing the need for virtual avatars [77].

Using these approaches a pervasive application can undergo significant testing before being deployed. However, there are several drawbacks to this approach. First, is the relative cost of creating emulators and simulators. Difficulty arises when attempting to build applications where simulators or emulators are not present, and the user is potentially faced with creating their own debugging platform. In this instance, less detailed testing techniques (such as Wizard-of-Oz) are more appropriate. Second, the benefit of the emulator/simulator is directly tied to the quality and diversity of information available. This is especially pertinent in simulations where the whole behavior of the prototype is being tested and not only the interface.

Wizard-of-Oz

Wizard-of-Oz techniques fake abilities within an application where a critical element is not present. In this way, the whole of the design can tested but without all the required pieces being present or functional. The means by which an element is faked is dependent upon the application. Wizard-of-Oz testing is less intensive testing that full simulation or emulation but is an appropriate testing technique early in design.

For GUI-based application it is common practice for Post-It notes to stand in for potential screens to test interactions [1]. In location based applications, such as Topiary [39], the user can click an on-screen map to generate position coordinates to the program. Similarly in Papier-Mache [40] the user can manually add and remove events detected by a vision-based system. In iCAP [50] system variables are manually edited to create events. For tangible interactions in dTools [36] program's statechart is manually operated by clicking on the state to enter next. Finally, DART [81] provides a WoZ interface for augmented reality applications where the operator can observe and manage events detected by the system.

2.3.6 Integration with Current Practice

This section describes tools and methods that integrate existing design practices into creating pervasive applications.

Foam Prototypes

A common practice in industrial design is using foam core to prototype the form of ideas. The Calder Toolkit [62] and IE5 [82] allow embedding of sensors within foam models. Calder contains wired and wireless components, such as buttons, joysticks, and knobs, that are controlled by a computer running C/C++ or Java, where the interactive program is written. IE5 allows the user to embed buttons on the foam that contain RFID tags. The user wears an RFID reader in a glove, which powers the tags and can read back the buttons ID numbers when they are pressed, thereby creating events for the system.

Sketching Interaction

Sketching is a fundamental activity for design and is a natural way to prototype interactive visual interfaces. Several tools have been created that allow direct sketching of graphical interfaces. Sketch Wizard [83] and Sketchify [37] enable sketching of graphical interfaces and help define transitions between those interfaces. STCtools [84] provides a similar ability but has the added benefit of using real buttons and sensors that can attached to a workspace where the interface is projected. Additionally, graphical interfaces can be annotated with d.note [85] which uses editors marks to indicate changes to interfaces, their behavior, and to make comments.

Different from these approaches, but also utilizing sketching, is Ex-A-Sketch [64] which allow teams to rapidly animate white board sketches. Sketches from a white board are captured using a camera which are imported and segmented into different objects using Adobe Photoshop^{TM}. Once the elements of the sketch are segmented they are projected back onto the white board and are manually animated using a control panel to prototype behaviors. For a more complete review of computational sketching and design, see [86].

Arduino Family

Most of the tools we have discussed are research attempts within the computing community to attract designers and novice users. However, there have been several home grown projects within the design world that incorporate computing elements. Most notable of these efforts are Arduino and Processing.

Arduino is an open source hardware and software platform for embedded computing [35]. Arduino has become a popular choice for creating interactive works because of its relatively low cost and its simplified programming language. The Arduino language is C-like, but previously complex operations, such as reading from an analog-to-digital converter, or communicating over a bus, have been simplified into single function calls. In this way Arduino has lowered the barrier to entry for many who are unfamiliar with embedded computing. Additional versions of the Arduino have been developed. The Lilypad Arduino [63] is a smaller form-factor Arduino that is designed to be sewn into fabrics. While the Lilypad is still programmed using the Arduino language its form allows easier integration into fashion design practices.

The Arduino approach of providing a simplified language was first used in Processing, a Java-like programming language that facilitates easy creation of interactive visual programs [34]. Processing simplified the drawing and movement of objects on screen in addition to providing access to keyboard and mouse events to make the program interactive.

On closer inspection, Arduino and Processing still retain many elements of their parent languages. Controlling an Arduino is still very similar to writing C, and creating a Processing program is still very similar to coding in Java. The key is these tools have lowered barriers to entry on specific actions that were previously difficult. Opening a window, and displaying and moving an object in Processing is a few lines of code, which when using traditional methods would be more involved. Additionally, Arduino allows code to be written, compiled, and downloaded onto the micro-processor within a single application; no programmers or additional tools are required. The success of these approaches suggests that entirely new approaches to integrating design and computing may not be necessary, but only that certain barriers be removed to provide easy access.

Field Observations & Storyboard

A unique representation is provided by Activity Designer [38] which allows activity models to be created from collected field data. The tool provides a structured method for activity observations to be imported into the tool to create various scenes which contain actions and situations. These scenes can be arranged in a storyboard metaphor to create interactive prototypes that can be simulated or run on target hardware. This tool is motivated by Activity Theory and Activity-Centered Design and maintains focus on the user and their activities throughout the design process.

2.4 Summary

As we discussed in Section 2.1, because pervasive computing experiences are closely coupled with computational material, the design of these systems must be interdisciplinary including engineering, design, and domain-experts in the fields where the system will be deployed. However, this close coupling presents challenges for teams due to these new computational materials (Sec. 2.2.1) and due to differences in design processes (Sec. 2.2.2). Given the importance of prototyping to the design process (Sec. 2.2.3) we examined the state of current pervasive computing design tools (Sec. 2.3) and present the results of that analysis here.

Examining the surveyed tools they fall into two large categories: either general programming languages aimed at novices, such as Scratch For Arduino [87] and ModKit [45], or custom tools targeted at a particular application domain, such as dTools [36], aCAPpella [46], CAMP [48], or Activity Designer [38]. In an interdisciplinary setting, either simplified programming tools, or domain specific ones are likely to fail. Members of the design team who are non-programmers will be forced to learn a programming language to engage in prototyping. Specifically, they will have a limited understanding of the "behaves like" aspects of prototyping [33]. While non-programmers do not require the same level of technical understanding as engineering team members, they still require a certain level of abstraction to understand why the prototype behaves the way it does. Having the proper level of abstraction about the operation of the prototype allows team members to reason about why a behavior did not work and attempt to debug the process. Additionally, they can be involved in the creative process instead of simply instructing the engineer about the desired behaviors. Tools tailored for a particular domain may work well, but cannot be used in all situations. Considering that pervasive computing applications span a wide domain from wearable computers to ambient spaces, it is undesirable for design teams to switch tools when working on different projects.

Proposal: In response to deficiencies in existing tools we propose to explore how to create a new tool that will enable interdisciplinary teams to collectively describe and implement the behavior of an interactive prototype. In particular, we believe that storyboarding can be used to describe the behavior of pervasive computing applications, in a medium that is accessible to all team members, and can contain sufficient formal properties to be used to implement prototypes. We advocate selecting storyboards over other mediums for three main reasons. First, by using storyboards in an interdisciplinary setting we are able to incorporate elements of sketching and drawing, which are fundamental aspects of design [1, p.96]. Drawing and sketching about a problem is reflexive activity that helps incrementally visualize and solve problems [13]. Second, storyboards are able to capture behavioral and temporal elements about a design [6] [1, p.296]. Describing behavior for interactive products can be difficult for designers; however, they often sketch and storyboard to resolve these issues [69, 88]. Finally, storyboards are accessible to members of an interdisciplinary team and are a format in which they can collectively reason about behavior [7]. We explore an implementation of electronic storyboards in the next chapter.

Chapter 3

Model-Driven Architecture for Pervasive Computing System

In the previous chapters we examined the needs of interdisciplinary teams when designing pervasive computing systems. From our analysis we argued that design teams require a shared view of the design space to effectively create pervasive systems and selected storyboards as our design front-end for developing such a tool. In this section we address Research Question 2 by outlining a model-driven approach to develop a new electronic storyboarding tool that implements the requirements described in previous sections. Electronic storyboards represent a domain-specific modeling language (DSML) for pervasive computing systems. Unique to our approach is the role of ambiguity in creating the pervasive system model. Users of electronic storyboards are not overly constrained when storyboarding their systems and are allowed to create designs that may not be "valid" when being transformed. This approach differs from traditional DSML development techniques which promotes rigor and formal meaning to ensure efficient model transformations. Our approach allows design teams to "make mistakes" which are interactively "corrected" during the model transformation.

In the remainder of this Chapter we outline related work in Section 3.1. In Section 3.2 describe our approach to implementing electronic storyboards and the challenges faced. In Section 3.3 we provide details of our solution and its operation on a sample storyboard. We summarize this work in Section 3.4[1].

[1]Portions of this chapter are based upon the author's work in [89] and are reproduced here under copyright permitted by the Association for Computing Machinery (ACM).

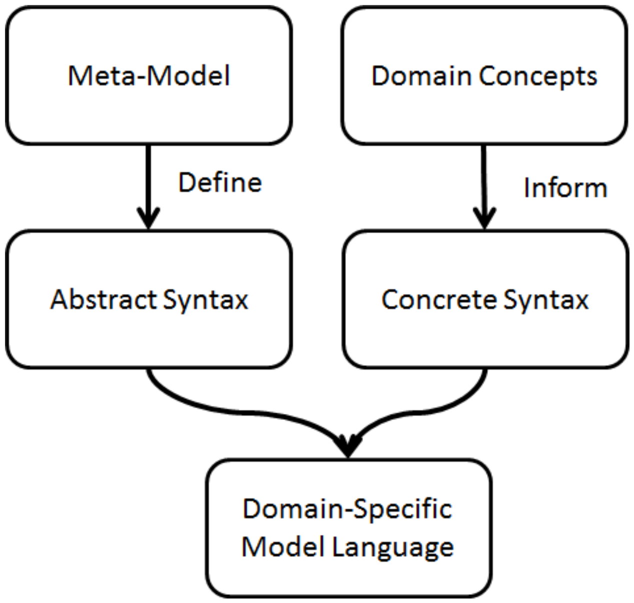


Figure 3.1: Elements of a domain-specific modeling language.

3.1 Related Work

In developing electronic storyboards we will adopt a model-driven engineering (MDE) approach to software development. MDE presumes that developing software is primarily a task of modeling some system in an appropriate language. This model is then transformed into other models through model-to-model transformations or into text and other artifacts through model-to-text or text-to-model transformations. The driving principles of MDE are to ensure properties of the system hold through these transformations and can be formally treated. Models of a system must conform to a meta-model that defines the set of allowable objects and relationships within that model [90].

Within Model-Driven Engineering, electronic storyboards can be viewed a domain-specific modeling language (DSML). DSMLs are different from general purpose languages as they are designed for applications within a single domain [91]. These targeted languages allow direct implementation of a model for that domain and enable automatic generation of other models or artifacts [92]. The argued benefits of DSMLs include easier maintenance of software, reusability, increased productivity, and better documentation [93, 94].

DSMLs come in two flavors: graphical or textual. Textual DSML are written languages widely recognized in computer science with common examples being Verilog, VHDL, and HTML [90]. More targeted textual languages are available that describe cyber-physical systems [95] and embedded systems [96]. Distinct from textual DSML are Visual DSMLs that use charts, graphs, and diagrams to express domain concepts [97]. As electronic storyboards are a Visual DSML we will focus the remainder of this section on their construction and application with MDE.

Visual DSMLs are defined by three elements: a concrete syntax, an abstract syntax, and grammar [90]. The influences of these elements are shown in Figure 3.1. The abstract syntax expresses the fundamental concepts in a domain and is generally implemented as a meta-model. The concrete syntax is the particular graphics and imagery that provide visual representation of the abstract syntax. Concrete syntax should be informed by the particular domain in which one is targeting so it is understandable to the domain end-user [98]. A grammar can be defined that expresses the valid "sentences" or models allowed by the abstract syntax and meta model. Within electronic storyboards, our abstract syntax is defined by the Electronic Storyboard Meta-Model in Figure 3.7. Our graphical syntax is the frames, images, and labels implemented within Eclipse GEF in Figure 3.9. The grammar that governs a model is generally expressed through the Object Constraint Language [99]. The issue of model validation, that is ensuring the model expressed by the DSML conforms to the meta-model, will be discussed in more detail later. The grammar governing electronic storyboard languages is more loose to allow ambiguity and foster discussion. This design choice does inhibit easy model transformation but we believe the benefits of fostering design discussion outweigh the challenges in the model transformation.

Developing DSMLs is recognized as a difficult task. Defining the abstract and concrete syntax requires collaboration between modeling and domain experts to ensure the two syntaxes are properly aligned and useful for the domain application. Research has examined accessible tools for defining DSMLs [100, 101, 102], enabling definition through demonstration [103], and approaches to focus primarily on the visual aspects of the language and automatically generate the abstract syntax [104, 105]. Sadly, the Eclipse Graphical Editor Framework used to implement electronic storyboards is not among the new accessible tools. At the time of starting the project, these new tools were not available or did not provide sufficient expressive power for our application.

A major initiative of model-driven engineering is to create artifacts that are "correct-by-construction" such that properties of models can be preserved through various transforms and modifications [106]. The benefit of "correct-by-construction" is most evident when developing source code or implementations of systems where properties like safety, reliability, and liveness must be retained. This focus expresses itself within DSML through the grammar rules or model-checking to ensure that a model expressed by the DSML is valid. If a model is invalid it is seen as an error and must be rectified by the user. Our approach in Electronic Storyboards is different. We intentionally allow teams to express an electronic storyboard that is ambiguous and difficult to perform the model transformation. Our assertion is the ambiguity in storyboards facilitates design discussion and concrete definition is not needed during the early design phases. However, as we will discuss in Section 3.3.4, we do desire the MDE properties to generate the state charts and source code from the electronic storyboard. Our shows there is a place for DSMLs during the early design stages and ambiguity can be valued instead of eliminated. The role of ambiguity to implement designs has received some recognition. The Epsilon Validation Language provides the ability for user queries during validation and can express different levels of "invalid" models through warning, critique, and error [107]. Additionally, work on uncertainty preserving model transformations [108, 18] has recognized that domain modeling is an iterative practice that utilizes refinement to remove uncertainty over time.

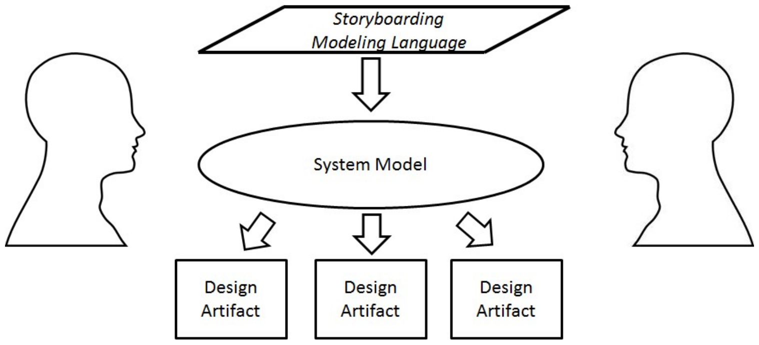


Figure 3.2: Electronic storyboards as a model-driven engineering process.

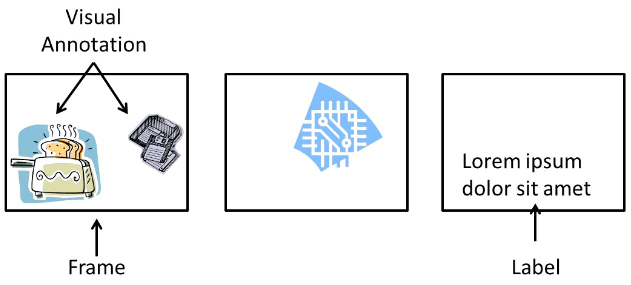


Figure 3.3: Elements of a storyboard.

In the next section we outline our approach to implementing electronic storyboard. We present our assumptions, the challenges inherent in using storyboards, and the models that drive our implementation.

3.2 Adopting a Model-Driven Approach

In this section we describe our model-driven approach to developing electronic storyboards. We this method for convenience and support as our primary function will be transforming storyboard information into a model of the underlying computing system. From that underlying model we will examine the set of artifacts that can be generated and how they may be useful to interdisciplinary design teams. Figure 3.2 outlines our approach. First, we develop a domain-specific modeling language (DSML) based on storyboarding that is accessible to both engineering and design practitioners. From that DSML, a model of a pervasive computing systems will be extracted based upon timed automata. Finally, the timed automata is transformed into design artifacts that are of interest to design teams. This section describes our assumptions in developing the electronic storyboarding DSML, challenges in creating that language, and the resulting meta-models that describe the pervasive system.

3.2.1 Definitions and Assumptions and Approach

An electronic storyboard is a software design tool that allows an interdisciplinary team to electronically draw and depict how a user, or group of users, interacts with some pervasive computing system. A storyboard typically contains a set of frames, with each frame containing textual and visual annotations as shown in Figure 3.3. Visual annotations encompass all the drawn elements of the storyboard, while textual annotations are the words and phrases placed in and around the frame.

A simplifying assumption made in this work is that the storyboard is drawn electronically using some computer application. By using an electronic medium, the drawings and markings on the storyboard can be recognized as independent objects, such as words and images, and not as a collection of individual strokes. This assumption removes the need for sketch recognition in the storyboard, and makes the challenge one of sensemaking and deriving high-level meaning. In the next section we outline some of the challenges faced by our approach.

3.2.2 Challenges Using Storyboards

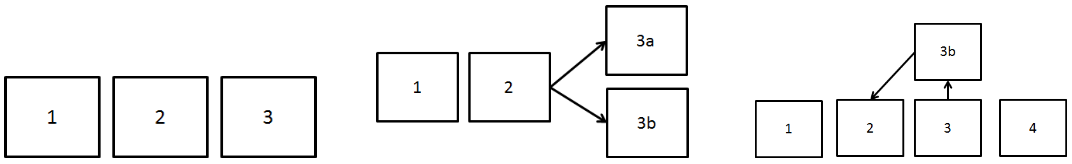
Software Specificity vs Storyboard Ambiguity

When using storyboards to describe pervasive computing systems, one of the most significant tradeoffs is between the current practice of storyboarding and the need to accurately capture the semantics of the pervasive computing system. Storyboards are often ambiguous and leave details to the reader. In practice this can be useful as ambiguity serves as a focal point for discussion between team members and moves the design process forward [1, p.117]. However, when describing pervasive computing systems, this ambiguity is a barrier to correct implementation of the intended system behavior. Prototypes of these systems often require assembling hardware and software components, where ambiguity in the implementation may create incorrect or undefined behaviors.

Our approach strikes a balance between the needs of the design team and the requirements of correct implementation through a combination of keyword “tags” applied to visual elements of the storyboard, and natural language processing of text throughout the storyboard. When creating a storyboard, the design team can “tag” visual elements within the storyboard to indicate that image contains important information. Our example tags are shown in Table 3.1 and allow the design team to indicate semantic information about a prototype’s behavior using State, Event, and Action tags, or indicate important contextual information using Person, Location, Temporal, and Context tags. This tagged information is supported by natural language processing (NLP) [109] of text within the storyboard. Any words, text, or labels contained within the storyboard are parsed using NLP to identify additional information regarding events, locations, or time. These NLP results supplement the information from the tags and allow the design team to incompletely describe a prototype, either intentionally or not, and thus enable the storyboard to retain some ambiguity. Given the inaccuracy of the natural language tools, their results are considered less authoritative than elements tagged by the

Keyword	Description	Example
Person	name of a person	Jimmy, Mom, Dad
Context	name of a context	Meeting, Outside
Location	physical location	At Home
Temporal	time interval	Later, Meanwhile
Event	triggering event	Push a button
Action	prototype's response	Display a Message
State	prototype state name	Idle, Waiting, Alert

Table 3.1: Supported tags for storyboard objects



(a) Linear Layout (b) Branching Layout (c) Looping Layout

Figure 3.4: Frame layouts to express different conditional behavior. The numbers in each frame indicate the order in which they would be "read" by the storyboarding tool. (a) shows linear storyboard frames that are read left to right. (b) allows for conditional behavior to branch away from the linear layout. (c) uses arrows to loop back on the storyboard.

design team. We resolve this issue by querying the user before any NLP information is accepted when creating the behavioral model.

Complexity of Behavior within a Storyboard

When using an electronic storyboard, the design team depicts the intended behavior of their prototype using frames, images, and text. Depending on the complexity of the intended behaviors, the storyboard can be rather large. Simple storyboards describe simple prototypes, but as the number of behaviors increases for the prototype, so does the complexity of the storyboard. While storyboards do not necessarily enable concise descriptions of a prototype we anticipate they do enable the description to be understood across disciplinary boundaries.

The complexity of the prototype is most readily apparent in the layout of the storyboard. Typical storyboards have a linear flow, meaning they are read left to right and top to bottom. When used to describe interactive behavior, the layout lends itself to expressing behaviors in a linear and causal order. An example of this linear layout in shown in Figure 3.4a. However, when developing interactive systems, there are often conditional or iterative behaviors that must be expressed. For example, a device should make a choice between two inputs, or should continue a behavior until some condition is met. Studying storyboarding more generally, Buxton commented that the complexity of the layout is directly related to the "learnability" of the underlying system [110].

Keeping the linear structure of a storyboard would make expressing these situations more difficult. To address this problem we have added arrows to connect frames with conditional events as shown in Figure 3.4b. In this way, the linear structure can be extended to exhibit branching behavior. These arrows can also be used to express looping

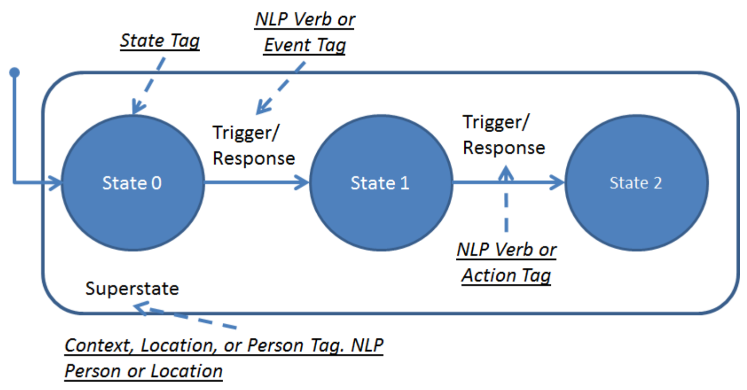


Figure 3.5: Mapping from storyboard objects onto a timed automaton

behavior where the storyboard returns back on itself as in Figure 3.4c. By augmenting the traditional structure of storyboarding we can allow design teams to express additional "computational" behaviors without sacrificing existing practice.

Mapping Storyboard and Model Semantics

Earlier in this section we discussed how information in storyboards can be expressed through keyword tags, natural language processing, and the layout of the storyboard. To enable these information sources to generate a behavioral model, their information must be mapped to a suitable model of computation. A "good" model of computation must have several properties: (1) support easy transformation of storyboard information to model information, (2) capture key prototype behaviors such as action, response, time, and context, and (3) support generation of design artifacts such as source code, diagrams, and other descriptions that help evaluate and implement a prototype. For our approach, we selected timed automata to represent the prototype's behavior within the storyboard. A timed automaton describes a system as a series of states, with each state having trigger conditions and responsive actions between states [111]. These models can be considered an extension of finite state machines as they allow transitions based upon time.

Timed automata are flexible and can be used to describe moderately complex systems [112]. Additionally, several of the keywords in Table 3.1 map directly to timed automaton concepts as shown in Figure 3.5. The keywords State, Event, and Action directly map to timed automaton states and transitions, whereas contextual information can be represented as a superstate that enables the timed automaton. Furthermore, timed automata can capture temporal phenomenon contained in storyboards. While frames are typically rendered in a linear order, their content can be highly variable with regard to temporal information. Timed automata are advantageous because transitions between states occur based upon an independent clock that is external of user input. In situations when there are temporal relationships between automata, for example some behavior must occur before another, we have adopted an interval algebra [113] that is used to specify how those automata should be ordered and executed. Finally, timed automata can

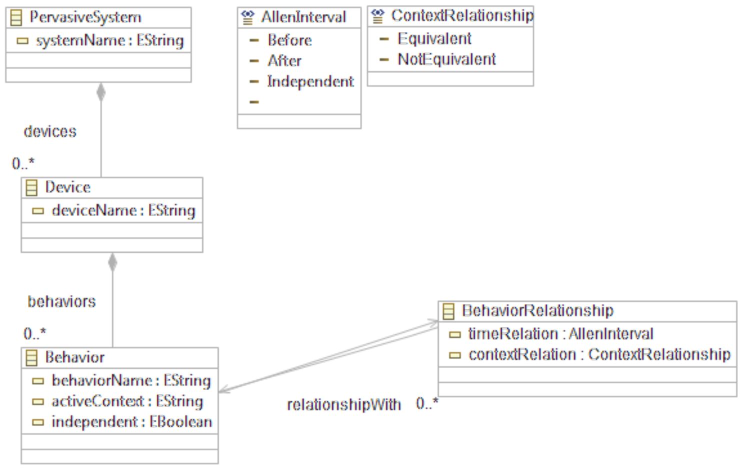


Figure 3.6: Meta-model of a pervasive computing system.

be used to automatically generate code [114, 115] that can facilitate implementing the prototype once the model has been formed.

3.2.3 Meta-Models of Pervasive Systems, Electronic Storyboards, and Timed Automata

Having outlined some of the challenges in this project, we now present more formal meta-models for pervasive computing systems, electronic storyboards, and timed automata. These meta-models define the fundamental concepts and relationships for each domain. We describe the meta-model in turn and outline our rationale for their construction.

Pervasive Computing System

Figure 3.6 presents our meta-model of pervasive computing systems. We define pervasive computing systems to be a collection of individual devices where the functionality of a device is defined by a set of behaviors. These behaviors encapsulate high-level interactions such as “responding to a message” or “reminding the user of an event”. Depending on the device in question it may have many behaviors that define its overall functionality. Each behavior does not operate in isolation but can have relationships with other behaviors based upon the context they operate under and any temporal dependencies between behaviors. Some behaviors might always be operational (e.g. receive a message), while others only at certain locations (e.g. silence phone while at the movies), or after other behaviors have occurred (e.g. upload my step count after running). We formalize these notions by defining two relationships between contexts - *equivalent*, *not equivalent* - and three temporal relationships between behaviors - *before*, *after*, and *independent*. As we will discuss in Section 3.3, the current context determines which behaviors are active. For behaviors with equivalent context they may all be active depending on their temporal relationship. Behaviors that are *independent* execute

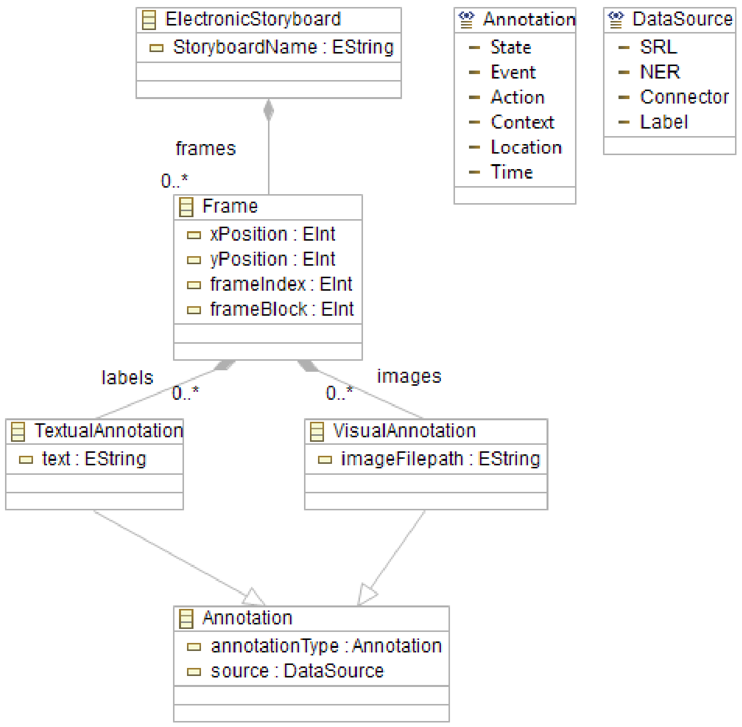


Figure 3.7: Electronic Storyboard Meta-Model

concurrently. If Behavior A is *before* Behavior B, A must complete before A can execute. The reverse condition holds for the *after* relationship.

Electronic Storyboard Meta-Model

An Electronic Storyboard contains a number of frames with each frame containing visual and textual annotations. Each annotation can have a particular source of information: a user-applied annotation, results from natural language processing using semantic role labeling (SRL) [116] or named entity recognition (NER), connector information assigned by the user, or direct text. We will examine how these information sources are used in later sections to facilitate model to model transformations.

Timed Automata Meta-Model

A timed automata is defined formally as $G = \{X, E, f, \Gamma, x_0, \mathbf{V}\}$ where X is a countable state space, E is a countable event set, f is a state transition function $f : X \times E \rightarrow X$, Γ is an active event function $\Gamma : X \rightarrow 2E$, x_0 is the initial state, and $\mathbf{V}$ is a clock structure that governs which active events are taken [111]. The meta-model, shown in Figure 3.8, captures these properties by defining states and transitions with triggering events and responsive actions. As we described in Section 3.2.2, timed automata are a suitable choice to represent the computational behaviors expressed in electronic storyboards. The tagged states, events, actions and natural language parsing within the storyboard help

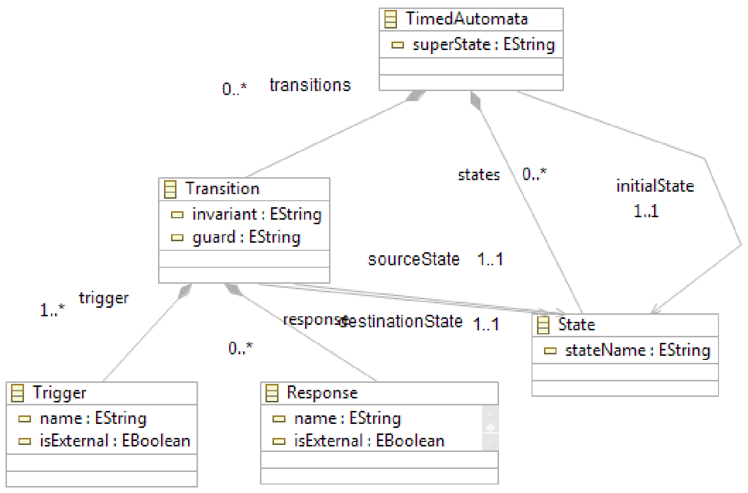


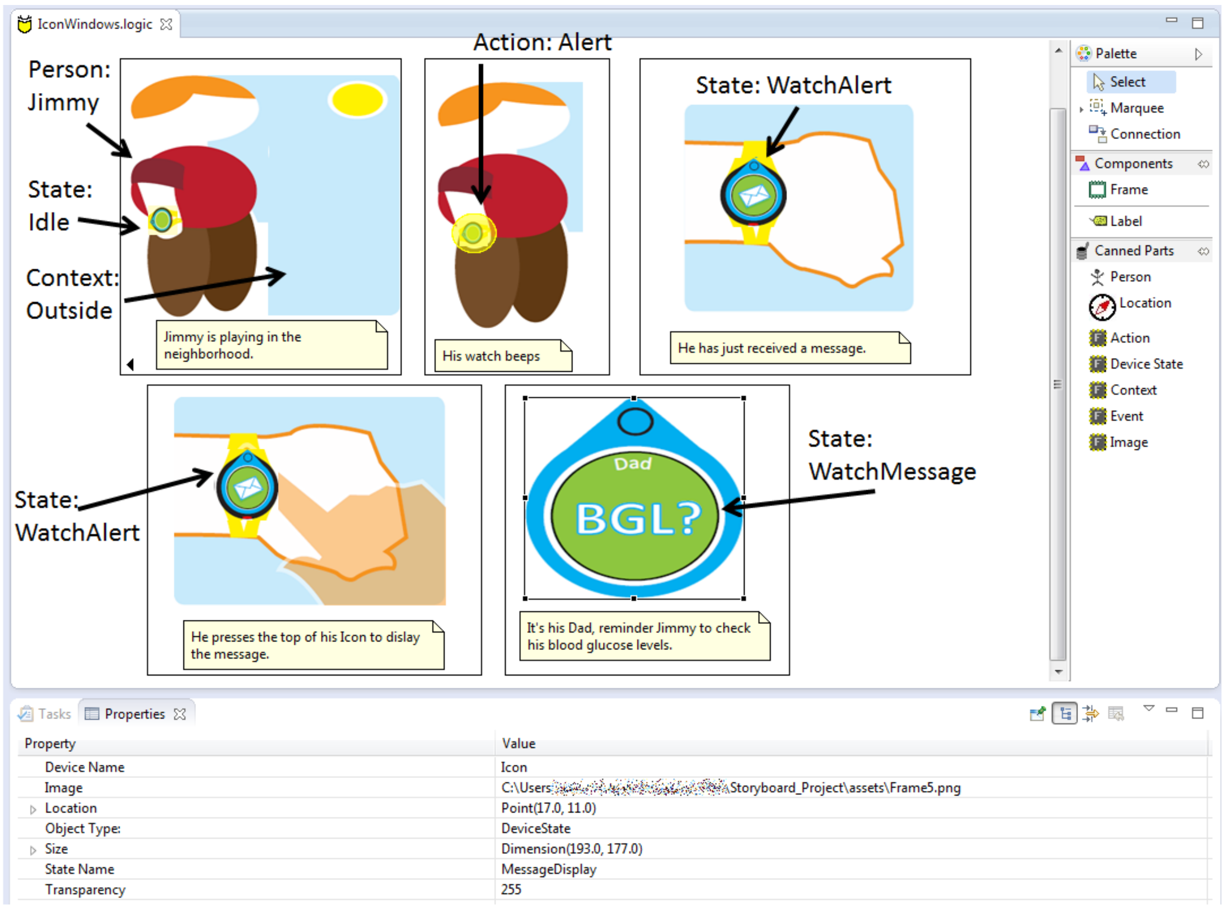
Figure 3.8: Timed Automaton Meta-Model

define the states (X) and events (E) of the automaton. As we will see in Section 3.3.1, the layout of the storyboard helps define the state transition function f that we extend with the actions identified through tagging.

3.2.4 Instantiating Electronic Storyboards in Eclipse GEF

An implementation of electronic storyboards has been created in Eclipse Graphical Editor Framework (GEF) that allows users to draw a storyboard using frames, images, and text [14]. GEF serves as a tool to implement our domain-specific modeling language. Images placed on the storyboard canvas can then be "tagged" using the set of keywords in Table 3.1 where each tag reflects a specific type of information. In addition to images, users can also place labels that contain arbitrary text, and frames to contain both the images and text and provide a structure to the storyboard. The design team can then "compile" the storyboard from within Eclipse and interact with the tool using the console.

Visual annotations and labels can be placed on the storyboard canvas by dragging and dropping elements from the palette shown on the right-hand side of Figure 3.9. These elements can be easily resized and moved in and out of frames. Specific information about each annotation, such as a Person's name or a particular Location can be modified in the Properties View below the canvas. In Figure 3.9, the Properties View shows the model information for a State tagged image, and allows the user to give the state a name, resize the image, and identify what device it belongs to. The attributes for each image change depending upon the keyword tag applied. For example, an image tagged as a Person could be given the attribute "John", or a tagged State as "Idle" or "Running". The attributes given to Context and Location objects have more significance as they help partition the storyboard into different sets of behavior. In addition to existing storyboard objects (frames, text, and images), we have added arrows that connect frames across the storyboard. These are used to indicate conditional behaviors in the storyboard that may not be directly indicated from the layout. A history of the design choices involved in developing Electronic Storyboards is outlined in Appendix B.



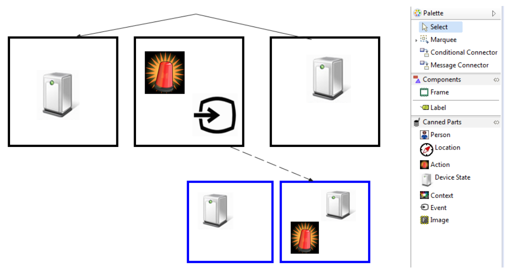
Property	Value
Device Name	Icon
Image	C:\Users[redacted]Storyboard_Project\assets\Frame5.png
Location	Point(17.0, 11.0)
Object Type:	DeviceState
Size	Dimension(193.0, 177.0)
State Name	MessageDisplay
Transparency	255

Figure 3.9: Screenshot of electronic storyboarding tool in Eclipse showing an example storyboard, palette, and properties view. Arrows indicate the type of tagged object and its value.

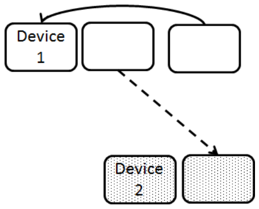
3.3 Model Transform from Electronic Storyboards to Timed Automata

Having defined the models in our tool we now outline the transformations required to move from electronic storyboards to timed automata. Storyboard compilation operates in three phases: Layout Analysis, Global Partition, and Local Synthesis as illustrated in Figure 3.10. Layout analysis transforms the structure of the storyboard into a graph that can be parsed by future transformations. Figure 3.10b provides a graphical illustration of this process. Layout analysis also ensures that connectors and device names are properly defined. Global partitioning takes the graph generated from Layout Analysis and partitions it into regions of different time and context as in Figure 3.10c. The assumption is that indications of time or context are different behaviors exhibited by the storyboarded device and will be complied separately. Finally, local synthesis converts each behavior into a timed automaton in Figure 3.10d.

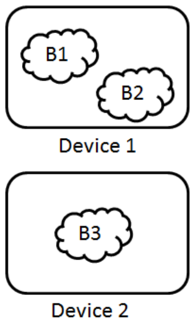
In this section, these transformations are presented at a high-level to maintain readability. Full descriptions of their implementation are presented in Appendix A.



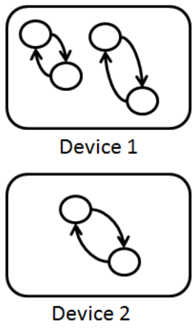
(a) An electronic storyboard showing looping and communication between two devices.



(b) Layout analysis

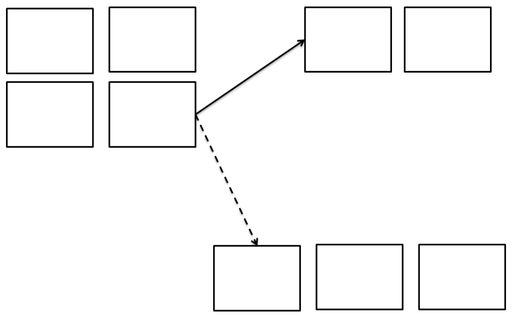


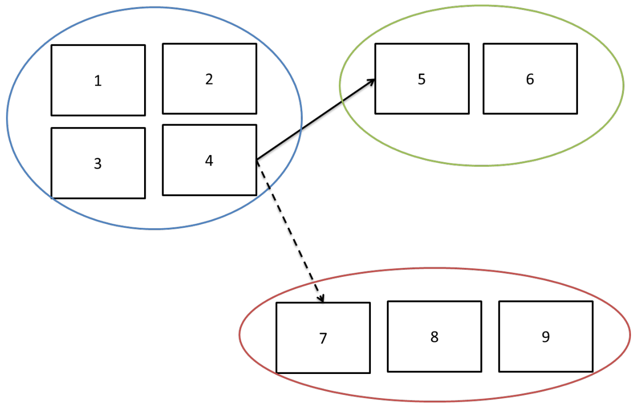
(c) Global partition



(d) Local synthesis

Figure 3.10: Example tool compilation process taking into account multiple behaviors and devices.


(a)



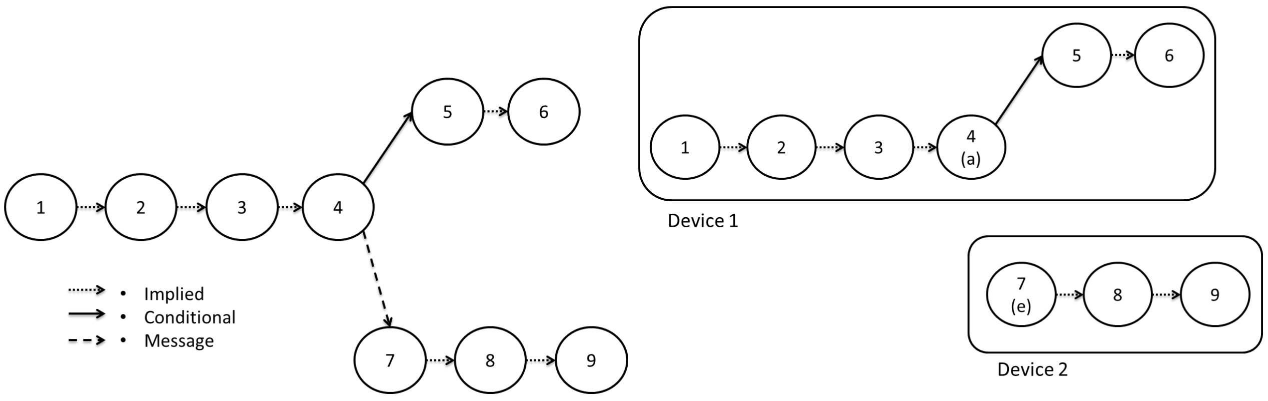
(b)

Figure 3.11: (a) Sample layout showing local, conditional, and message connections. (b) Frames grouped by geometric proximity.

3.3.1 Layout Analysis

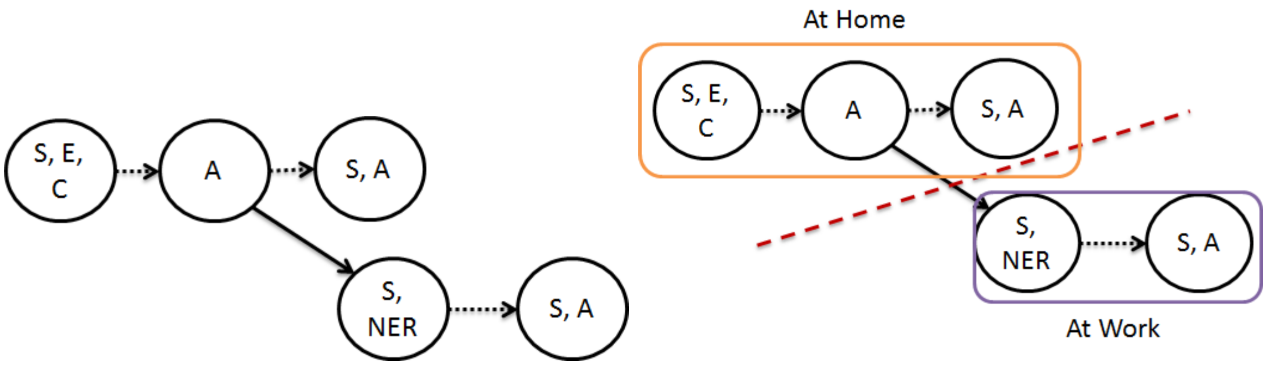
The first step of compilation is to extract layout information from the storyboard. Figure 3.11a shows a sample storyboard with three groups of frames. The solid connector indicates a conditional branch between the groups, while the dashed connector shows that a message is being passed between two devices in that group. The first pass in layout analysis groups frames by geometric distance. Figure A.2b identified three frame groups that were found.

After identifying frame group each frame is enumerated and the group is "read" left to right and converted into graphs as shown in Figure 3.12a. Each node in the graph is a particular frame and each node is connected by an arc that identified what layout relationship the frames have to each other. Frames can be connected by "implied" connections through physical geometry or explicit conditional and message connectors available through the user interface. The final stage in layout analysis is the only stage where the layout is constrained. The State annotations in each group of frames is examined to determine if multiple devices are references. Conditional connectors may only connect groups that reference the same device, and message connectors may only be used to reference groups with different devices. In



(a) (b)

Figure 3.12: (a) Resulting graph from Layout Analysis. (b) Cutting message connectors. Inserting events (e) and actions (a) into the form connecting frames.



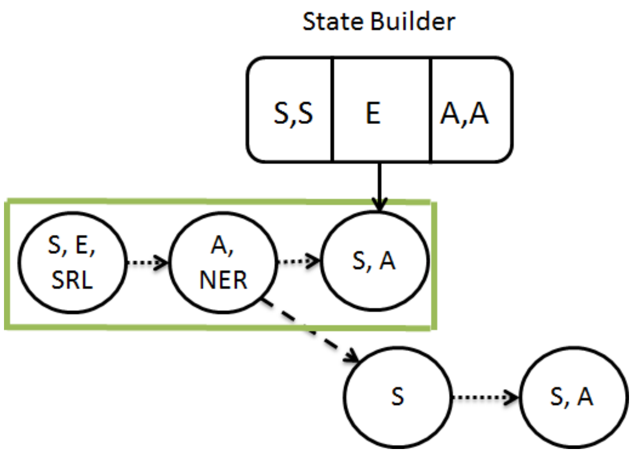
(a) (b)

Figure 3.13: Global partitioning of a storyboard graph. State, Event, Action, and Context annotations are indicated with S, E, A, C respectively. NER indicates natural language information from Named Entity Recognition.

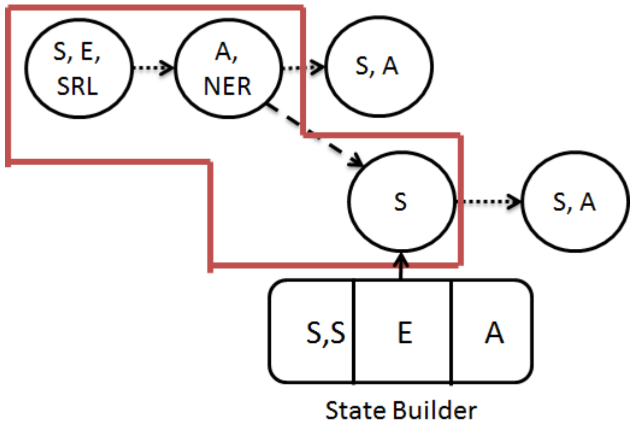
Figure 3.12b the graph is partitioned based up the difference devices it references. The message connector is removed and replaced with a sending action in the originating frames and a receiving event in the destination frame.

3.3.2 Global Partition

Once Layout Analysis has completed, the graph for each device is examined for Context and Time information. Recall from Section 3.2.2, context and time information can be found from Context, Temporal, or Location tags, or from location results from natural language processing. Global partitioning operates on this context information to partition the graph into sub-graphs of different time and context. Following every path down the graph a partition is created when a new context is identified. After partitioning if multiple contexts are available attempts will be made to re-merge equivalent contexts into a single graph. Otherwise each graph will be treated as an individual behavior of the device.



(a) Multiple Action annotations.



(b) Complete transition.

Figure 3.14: Local Synthesis parsing two branches.

As an example of this process, the graph in Figure 3.13a contains five nodes with varying types of information. User-applied State, Event, Action and Context annotations are indicated with S, E, A, and C, respectively while natural language information from Named Entity Recognition is indicated with NER. In this example 'C' contains the context "At Home" while the location "At Work" is inferred from NER. During global partitioning the user will be queried to determine whether "At Home" and "At Work" are different contexts. Assuming the user indicates they are different, the graph will be partitioned into different sub-graphs as shown in Figure 3.13b.

Currently, the tool can identify that some information is related to context or time, but it cannot distinguish between different contextual information. For example, two contexts "at home" and "at work" that are tagged by the design team are easily recognized, but the tool itself has no means to distinguish these contexts and requires the design team to differentiate between the two. A semantic understanding of these contexts could be accomplished through tools that enable novice users to define contexts of interest [46], or existing architectures to recognize and disseminate context [117]. However, an implementation of these approaches is beyond the scope of this work. Regarding temporal information, our approach does determine relationships between time intervals by using interval algebra [113] to provide a formal definition of temporal ordering.

3.3.3 Local Synthesis

After the storyboard has been partitioned into sets of similar time and context, each set is analyzed to build a timed automaton. The graph is searched for user-applied State, Event, and Action annotations which correspond to states of the automaton, the triggering events between states, and the actions taken by the system. With two states, it is known that a transition has occurred, but the triggers and responses may be unknown. In the absence of tagged objects, the textual annotations within the storyboard are queried based upon SRL and NER parsing results. If NLP information

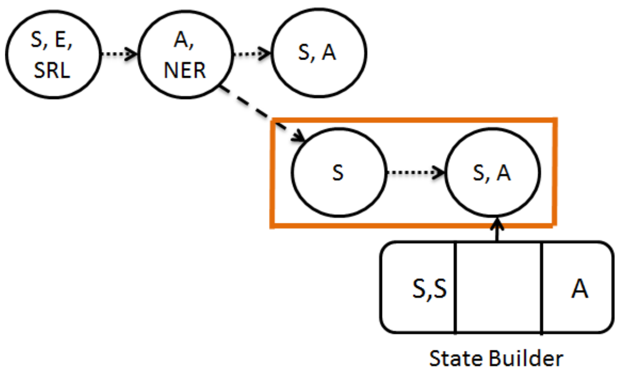


Figure 3.15: Parsing final branch of local synthesis with missing Event annotation.

```
switch(currentState)
{
  [for (state:State | ta.states)]
  case [state.stateName/]:
  [for(t:Transition | ta.transitions->select(tran:Transition | tran.sourceState=state))]
  if([(t.trigger.name)/]()==true) {
      [(t.response.name)/]();
      nextState=[t.destinationState.stateName/];}
  [/for]
  break;
  [/for]
}
```

Listing 3.1: Implementing Arduino state machine logic in Acceleo. Each state in the timed automata is implemented as a series of case statement and inbound and outbound transitions are automatically implemented.

is not available, or the user does not select any NLP results, the user will be asked to manually specify the missing information. Additionally, if multiple tagged events and actions or available, the user will be asked to specify which events and actions cause the transition.

As shown in Figures 3.14 and 3.15, a State Builder follows the graph collecting these annotations. The State Builder stops when when two State annotations are found and attempts to build a transition. In Figure 3.14a the Builder reaches the end of the graph with two State, one Event, and two Action annotations. To help resolve the multiple Action annotations the user will be queried to determine which Action is caused by the lone Event. In Figure 3.14b the Builder has continued down a branch to build another transition. In Figure 3.15 the final transition is found but with a missing Event annotation. With no natural language information within this branch, the user will be queried to describe the missing Event.

3.3.4 Resolving Behaviors and Generating Design Artifacts

The previous section One the main benefits of adopting timed automata was they could be synthesized into various artifacts that might be useful for a design team. In this section we outline how timed automata can generate source code, state charts, and textual descriptions. These artifacts will be used during the user study described in Chapter 4.

```
static int pThreadsilence(struct pt* thread){
  enum State {MovieIdle, Silenced};
  static State currentState=MovieIdle;
  static State nextState;
  PT_BEGIN(thread); //Begin thread body
  while(1){
    if(AttheMovies()==true){ //is context active?
      switch(currentState)
      {
      case MovieIdle:
        if(ReceiveMessage()==true)
        {
          SilenceMessage();
          nextState=Silenced;
        }
        break;
      case Silenced:
        break;
      }
    }
    currentState=nextState;
    PT_YIELD(thread); //yield to others
  }
  PT_END(thread);} //End thread body
```
Listing 3.2: Automatically generated context-enabled behavior implemented as protothread.

Resolving Device Behaviors

After partitioning and synthesis, the behaviors extracted from the electronic storyboard are implemented as a timed automaton as shown in Figure 3.10d. Before these behaviors/automata can be generated into source code, the contextual and temporal relationships between the behaviors must be resolved. As discussed in Section 3.2, each behavior can have contextual and temporal relationships with one other. Contextual relationships are defined as *equivalent* or not *equivalent*. Determining whether two contexts are equivalent can be difficult as their relationship may change based upon the application. For example, in the case of a watch that automatically silences calls during personal situations, being “in a meeting” or “dinner with friends” are different situations but are the same context for this application. Given these difficulties the tool asks the user to manually resolve these relationships through a text console. The current approach supports three temporal relationships: *before*, *after*, and *independent*. *Before* and *after* are reflexive relationships that indicates one behavior must complete for the other one to execute. Behaviors may also be independent where they can execute without regard for one another.

Generating Design Artifacts

Source Code: After resolving relationships between behaviors each device in the storyboard can be implemented in the Arduino language. Using the Acceleo model-to-text language [118], the state transition logical and function calls for each automata can be automatically implemented using Listing 3.1. The program reads in a timed automata model and generates the required transition logic and functional stubs in Arduino. Any text within brackets is part of the

```
void coffeeMessageRX(){
  OSCMessage msg("/COFFEESTARTMESSAGE");

  //use msg.add() to add custom payload data

  Udp.beginPacket(outIp, outPort);
  msg.send(Udp); // send the bytes to the stream
  Udp.endPacket(); // mark the end of the OSC Packet
  msg.empty(); // free space occupied by message
  return;}
```

Listing 3.3: Sending OSC message from Alarm Clock to Coffee Maker.

```
boolean coffeeStartMessage(){
if(coffeeStartMessageMSG==NULL)
      return false;
//message has been received
else {
  //do something (if any) with message contents

  coffeeStartMessageMSG->empty();
  coffeeStartMessageMSG=NULL;

  return true;}}
```

Listing 3.4: Generated message receiver on Coffee Maker.

Acceleo language and other text is directly printed out. This enables easy code generation using the timed automata derived from compilation. For a device where more than one behavior is present each behavior is implemented as a protothread to enable concurrency [119]. Protothreads are light-weight threads implemented as co-routines in C. Utilizing protothreads and Acceleo the state transition logic of a timed automata can be automatically generated as in Listing 3.2. The code in Listing 3.2 partially implements a behavior of a smart watch that silences inbound messages if the wearer is at the movies. While Acceleo can implement the timed automata's transition logic it cannot directly implement individual functions, such as checking for a message or where the user is located. Overall this reduces the amount of code required to implement the device but does not eliminate writing code entirely.

As an additional benefit for pervasive systems where multiple devices are communicating, Acceleo can be used to automatically generate message passing structures. Open Sound Control (OSC) [120] is used to encapsulate messages and use the UDP library in Arduino to send packets over Ethernet or Wifi. Each message connector in the storyboard must provide a name for the information that it is transmitting. This name is used as the address pattern in the OSC message to uniquely identify the message between sender and receiver. Listing 3.3 and 3.4 are derived from an example where an alarm clock checks whether the user is awake and asks whether they would like coffee. Depending on the user's response a message is sent to the coffee maker. Listing 3.3 automatically implements the code required to send a message between the two devices and allows the user to add custom data to the payload. Program 3.4 receives the message and routes the message object to the corresponding function where the payload is unpacked.

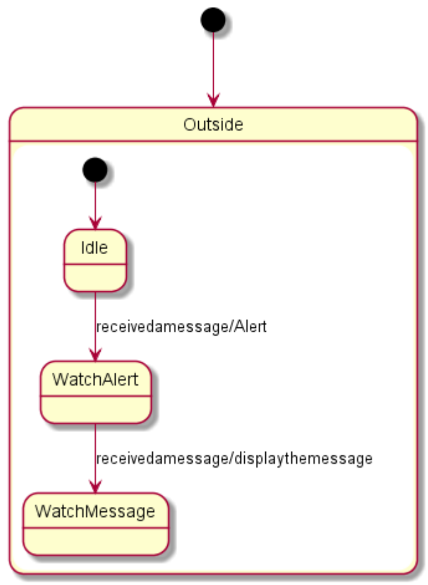


Figure 3.16: State chart generated from storyboard

State Charts and Textual Descriptions: While source code is helpful in implementing the prototype, it may not be an accessible representation for all users of electronic storyboards. Fortunately, the timed automaton formalism can be expressed in several ways. We use the PlantUML [121] library to create graphical statecharts as in Figure 3.16. Additional textual descriptions can be generated that express the state chart in words rather than visually. In the user study in Chapter 4 we will examine the utility of these representations.

3.3.5 ICON: Storyboard

This section showcases an example of how to use electronic storyboards to synthesize a timed automaton. We have developed a Java-based proof-of-concept tool in Eclipse GEF that implements the information extraction process described in the previous section. The tool reads an electronic storyboard and interacts with the user to resolve ambiguous or missing information in the storyboard. The example storyboard, as shown in Figure 3.9, is taken from an interdisciplinary product design course [3] and has been re-created by the authors with keywords to describe the prototype's behavior and provide a proof-of-concept for our approach. No other changes have been made to the original storyboard.

While this section only examines a single example storyboard, the example has been intentionally chosen as it highlights many common difficulties encountered when synthesizing electronic storyboards. Based upon our experiences working with college-level product design teams and recent work with middle school students, most storyboards will be incompletely tagged and fall under the same class of storyboard as our example here. It is important in this example that the tags applied to the storyboard do not fully specify the prototype. As we will see, there are missing

Frame	Tags	Semantic Role Labeling	Named Entity
1	Person:Jimmy, Context:Outside, State:Idle	A0: Jimmy V: playing LOC:in the neighborhood	Jimmy:Per
2	Action:Alert	—–	—–
3	State:WatchAlert	A0: He V: received A1: a message	—–
4	State:WatchAlert	A0: He V: presses A1: the top of his Icon	—–
5	State:WatchMessage	A0: Jimmy V: check A1: his blood glucose levels	Jimmy:Per

Table 3.2: Information extraction from an example storyboard (V=verb, A0=direct object, A1=indirect object, LOC=location, TMP=temporal, PER=person)

"event" tags in the first three frames, and no tags for "event" or "action" in the later frames. This forces our tool to rely on natural language processing and to query the user for missing information. As this dissertation is a study of the feasibility of using electronic storyboards, and this example is representative of many common storyboards, our methodology must show that it can address these types of storyboards to be a viable design tool.

The example storyboard in Figure 3.9 shows a child interacting with a smart watch. The storyboard illustrates how a father and son can communicate to keep up to date on the son's blood glucose levels. The parts of the storyboard shown illustrate the response of the watch when it receives a message and how the son can push a button to display and read the message.

In Figure 3.9, tagged elements of the storyboard are indicated by arrows pointing to different visual annotations. In the first frame there are three tagged elements. The image of the child is tagged as a Person and given the name Jimmy. The picture of the watch is tagged as a State of a device and assigned the name Idle. Finally, the first image with the sun is tagged as a Context and assigned the name Outside. Once these elements are tagged, their information persists across the storyboard. Thus in the second frame, the watch is known to be in an Idle state without having to re-tag the visual annotation. In addition to tagged storyboard elements, the results of the NLP parsing are shown in Table 3.2. For the SRL and NER results, V indicates a verb, A0 a direct object of the verb, A1 an indirect object of the verb, TMP a temporal modifier, and LOC a location.

As described in the previous section, the storyboard is automatically converted into a timed automaton in two phases. First, the storyboard is partitioned into sets of frames that occur under the same context and during the same time interval. After the frames have been partitioned, each set of frames is parsed to isolate behavior about the prototype. Each frame is searched for states of the prototype, events that it responds to, or actions that the prototype performs. For our example storyboard, the timed automaton in Figure 3.17 is generated by the process. We illustrate how that automaton is created in the remainder of this section.

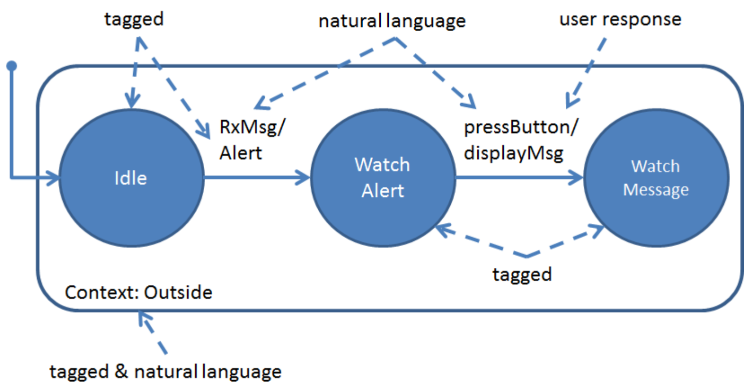


Figure 3.17: Timed automaton derived from example storyboard. Information sources are indicated with dashed arrows.

Partitioning Based Upon Time and Context

Beginning with the first frame, the storyboarding tool searches for context and time information. Initially the tool does not have any understanding of time or context, but adopts the first meaning that it finds. From that point forward, new time intervals or contexts are compared with the current to see if they are similar. Using the information in Table 3.2, the tool searches for frames that provide information regarding time or context. Time information is found from any tagged Time keywords, or any NLP result with the TMP tag. Similarly, context information is found from any Context, Location, or Person tags or any NLP result with the LOC tag (indicating location).

The Context "Outside" is created as the initial context, as it is found from tags in Frame 1. However, within the same frame, a location "in the neighborhood" is found from the NER results. Presently, our method cannot determine the difference between contexts based solely upon name, so the user is queried via the console to determine if they are different. For this storyboard containing the contexts "outside" and "in the neighborhood" the user would respond that they are equivalent contexts so the tool continues through the storyboard. As no new contexts are encountered through the remainder of the storyboard, all the behaviors within the storyboard are assumed to occur under the context "outside". This is represented by the superstate in Figure 3.17 that contains the whole automaton.

Building Local Behavior

After partitioning the storyboard, the tool scans each frame for State, Event, or Action information until one of two stop conditions is reached: two states have been found, or an event and action have been found. Each condition indicates that a change in behavior has occurred. With two states, it is known a transition has occurred but the triggers and responses may be unknown. With an event and action the transition is described, but its originating and next states are unknown.

Returning to the example storyboard in Figure 3.9, the tool reads Frames 1 to 3 and encounters two states and an action. Frame 1 shows the smart watch in the Idle state, Frame 2 shows the Action alert, and Frame 3 provides a new state called WatchAlert. Currently, two states are known (Idle and WatchAlert) along with the Action "Alert", which is the watch's response to the state change, but the trigger of the state change is unknown. With no tagged information to guide it, the tool asks the user for the triggering event. Since the action "Alert" is known the tools asks "Does the following statement cause the action Alert?" This question is posed for each SRL result in Table 3.2 and the user is asked to respond 'yes' or 'no'. For the example storyboard, the user is asked whether "playing in the neighborhood" or "received a message" caused the Action Alert. The user would respond that "receive a message" is the correct trigger. Using this process, the tool has found that the watch moves from State Idle to State WatchAlert when "received a message" occurs and should respond with an Action called Alert. This information is represented in the timed automaton in Figure 3.17 as a transition between the two states Idle and WatchAlert. After creating this behavior, the tool continues parsing the storyboard.

Beginning in Frame 4 the tool encounters State WatchAlert and then State WatchMessage in Frame 5. In contrast to the earlier frames, there are no tagged events or actions to indicate what causes the transition between these states. The tool attempts to resolve this issue by asking "Does the following event cause the system to transition from WatchAlert to WatchMessage?" using the SRL results in Table 3.2. Thus, the user would be asked whether "presses the top of his Icon" or "check his blood glucose levels" is the triggering event of the transition. Here the user responds that "presses the top of his Icon" causes the state transition. However, the transition cannot be completed as a responsive action is still missing. The behavior implied by the storyboard is that the message should be displayed after the button is pressed. In this situation the tool would continue to ask the user if the remaining SRL result, "check his blood glucose levels", is the responsive action. As this is not the expected behavior the user would decline these result. Having exhausted all information resources, the tool will ask the user to manually specify the action. The user could then manually type a response such as "display the message" on the console. This final behavior can now be added to the timed automaton in Figure 3.17 as a transition between WatchAlert and WatchMessage, caused by pressing the button, and the user supplied response.

After reading Frame 5 the parsing of the storyboard is finished. The automaton in Figure 3.17 represents the behavioral model produced by this process. Additionally, source code can be generated from the automaton. Listing 3.5 shows a portion of generated code that shows the transition logic for the automaton using the Arduino programming language [122]. The code initially checks to if "outside" is the current context and then executes the logic defined by the automaton.

```
State currentState=INITIALSTATE;
State nextState;
void loop(){
  if(isOutside()==true){
    currentState=Idle;

    switch(currentState){
    case Idle:
      if(receivedAMessage()==true){
        alert();
        nextState=WatchAlert;
      }
      break;
    case WatchAlert:
      if(pressesTheTopOfHisIcon()==true){
        displayTheMessage();
        nextState=WatchMessage;
      }
      break;
    case WatchMessage:
      break;
    }
  }
  currentState=nextState;
}
```

Listing 3.5: Arduino code created from the timed automaton in Figure 3.17

3.4 Summary

In this chapter we have outlined a model-driven approach to the development of electronic storyboards. The work in this chapter addresses RQ #2 by showing how tagging, natural language processing, and layout analysis can be used to generate timed automata of storyboarded systems. The proposed solution is not the only possible approach but reflects the challenges identified in Section 3.2.2. In the remainder of this dissertation we propose models to evaluate our tool in Chapter 4 and report on the tool's efficacy in Chapter 5.

Chapter 4

Evaluating Electronic Storyboards As Boundary Objects

4.1 Introduction

Examining the efficacy of electronic storyboards is challenging. As a socio-technological, system many factors can influence the "performance" of teams during the study including participant backgrounds and interests, interaction modalities, and tool responsiveness. To address this concern we will frame an electronic storyboard as a boundary object [123] within the context of computer-supported collaborative work. Boundary objects are "shared information artifacts" that support exchange and transmission of information between different domains and communities of practice.

Boundary objects have been studied extensively within Computer Supported Collaborative Work (CSCW). Medical orders [124], data bases [125], spreadsheets, architectural diagrams, and timelines [126] have all been identified as boundary objects and have been used to examine how groups utilize these objects to conduct collaborative and cooperative work. The definition of what constitutes a boundary object is broad and the concept has been expanded in areas of boundary negotiation [127], assemblages [124], cultural probes [128], and boundary zones [129] to describe additional forms of work and collaboration. Furthermore, boundary objects need not be static objects but can be items that are built during a process or appropriated from other disciplines [130].

By modeling electronic storyboards as boundary objects, we can leverage established boundary object properties and existing usability approaches to evaluate electronic storyboards and conduct a user study. However, because the notion of boundary objects is very broad, our expected use cases must be properly established. We address these two issues in the remainder of this section. In Section 4.2 we outline a model of design for pervasive computing based upon three properties: *computation*, *user interaction*, and *physicality*. Leveraging boundary objects, we discuss how those properties are expressed in collaborative design. From this model we outline a user study in Section 4.3 to evaluate electronic storyboards as a collaborative design tool. We summarize this work in Section 4.4 and look towards the study results in the next chapter.

4.2 Modeling Product Design in Pervasive Computing

In this section we provide a model of how design occurs within pervasive computing. We describe design as balancing of constraints between computation, user interaction, and physicality mediated by boundary objects. Section 4.2.1 describes our model of pervasive systems and Section 4.2.2 outlines how boundary objects are used in collaboration by design teams.

4.2.1 Pervasive Computing as Balancing Computation, User Interaction, and Physicality

For our boundary object model to be useful we must first describe the potential design in which interdisciplinary teams may operate. As described in Section 2.1, pervasive computing systems describe a range of systems from wearable devices to cyber-physical systems. Furthermore, pervasive systems may be composed of multiple devices and have multiple potential users. To capture this large variety of systems we outline three properties, *computation*, *physicality*, and *interaction*. In our model, *physicality* embodies the physical nature of the computing system and includes its size, weight, look, choice of material, and tangible interaction with the user. *Interaction* describes the actions, times, places, and context involving the user(s) that are of interest to the system. Finally, *computation* represents the sensing and logic that enact a pervasive computing system's behavior.

These properties are not meant to be exhaustive, as there are other properties such as cost, life cycle, usability...etc. that may be considered, but will be used as a framework to describe how design proceeds for pervasive systems. Given these properties, we propose that design is a balance of constraints [5] and when designing pervasive computing systems that these properties are not isolated, but influence each other given certain design choices. For example, the physical size of a system may restrict the availability of sensing and computing resources simply because the components cannot be contained with the device. Likewise the size of the system is dictated by the needs of the user and the affordances of the system's size - a wearable computer is expected to meet particular size and weight requirements [131] that would not be required of a smart display or ambient interface. Visualizing these relationships in Figure 4.1 a pervasive system would balance these three properties.

In the next section we explore the mechanism through which teams express these properties and exchange design ideas. We describe collaborative design through the use of boundary objects and discuss how electronic storyboards could facilitate discussion of design ideas.

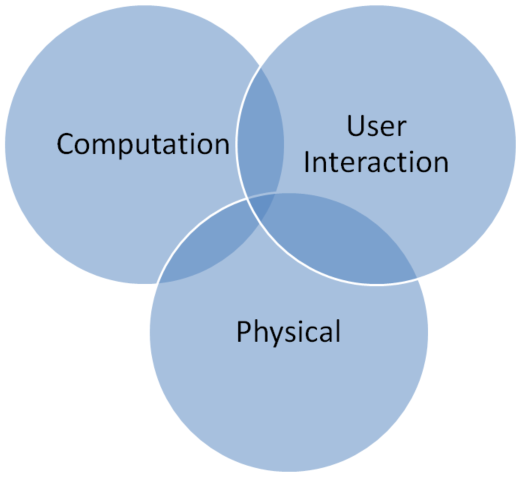


Figure 4.1: Pervasive Computing as a Balance of Properties and Constraints

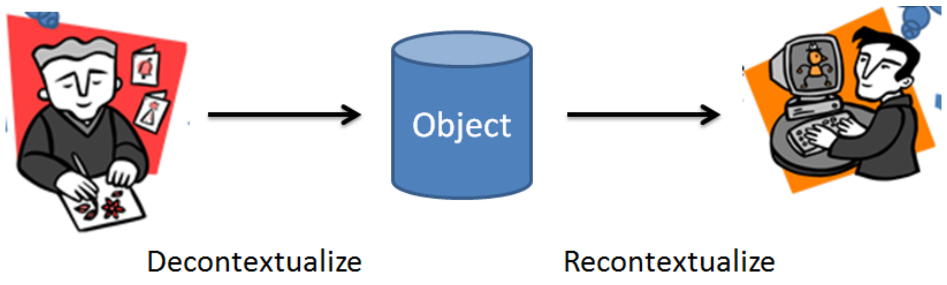


Figure 4.2: Using Boundary Objects

4.2.2 Boundary Objects in Collaborative Work and Product Design

In the previous section we outlined our framing of interdisciplinary design for pervasive computing products. To gain a better understanding of the dynamics of interdisciplinary teams while using electronic storyboards, we model pervasive design as centered around “boundary objects”. Boundary objects are shared information artifacts that enable transmission of information between different disciplines and communities of practice [123]. As initially envisioned, boundary objects are physical artifacts that “reside between social worlds (or communities of practice) where [information] is ill-structured”, can be used in domain-specific as well as interdisciplinary work, and groups working together with these objects alternate between these uses [132]. As illustrated in Figure 4.2, in each discipline or community-of-practice, specialized information is “decontextualized” into the boundary object. On the other side that information is “recontextualized” within a new domain. The value of a boundary object is based upon how effectively it facilitates this decontextualization and recontextualization process [133].

While the concept is malleable to describe many situations it is best employed when: (1) the objects in use reside between social worlds, (2) the object is useful for both interdisciplinary and non-interdisciplinary work, and (3) the object allows groups to collaborate without consensus [132]. This final property, the ability to work without consensus, shields individual participants from the full knowledge of each discipline involved, allowing them to work together without fully understanding all the discipline specific details.

In collaborative design, the information shared through the object and across boundaries is “pragmatic” [134]. Pragmatic knowledge helps participants understand knowledge differences and the consequences of decisions to each discipline. Good boundary objects expose knowledge at the correct level of abstraction and with sufficient detail to enable teams to “learn across a boundary of practice” but “operate in condition of partial ignorance” which is sufficient to work together and accomplish tasks [135]. These objects help teams span the “conceptual dimension” of collaboration and promote discussion at the boundaries of practice [136]. These discussions at the boundaries can result in “breakdowns” where the design must be re-evaluated in the light of additional constraints [27]. These breakdowns cause iteration on design ideas that are essential to innovation in the design process [32, 137, 138].

Electronic Storyboards as Boundary Objects

In this section we describe in more detail how electronic storyboards, as opposed to traditional storyboards [139, 140], function as boundary objects. In describing boundary objects, Star writes about the standardized collection methods employed by Grinnell [123, p.406]:

> *The method protocols themselves, and the injunctions implied, are a record not only of the kinds of information Grinnell needed to capture for his theoretical developments, but of the conflicts between the various participating worlds. In a sense, each protocol is a record of the process of reconciliation.*

The key method in coordination within Grinnell’s museum was a check sheet that aligned the practices of professional biologists and local trappers. For electronic storyboards, we too have a “check sheet” that are the models and meta-models defined in Chapter 3. These models represent the required information to describe pervasive computing systems and provide a “standardized method” of reconciling disciplinary information between design and engineering participants. Examining the meta-models more closely, the key information that is required is the *logical ordering* and *description of the events, actions, and states* of the underlying computing system. Selecting this information is intentional as we believe it provides the correct level of abstraction to enable collaboration between design and engineering. To allow this information to be expressed more easily we utilize storyboarding to focus on the user experience as a common touch point for expressing design and computational concerns. As we will see in the next chapter, as participants storyboard, apply tags, and respond to queries, they express *computational*, *user experience*, and *physical* concerns about the system under design.

While it is difficult to formalize more rigorously as our participants are from multiple disciplines, in describing the user experience, participants are provided an opportunity for their disciplinary knowledge to impact the definition of the pervasive system. Our selection of storyboard is critical to that expression as it attempts to balance the relationship between the two disciplines. Other boundary objects, such a programming languages, diagramming tools, foam models, or renderings could be used, however we believe alternative form do not allow a balance of concerns and would not enable "equitable relationship" between participants [141].

While we have identified electronic storyboards as boundary objects, electronic storyboards offer a distinct advantage over traditional boundary objects as they can serve as "design critics". Opposed to traditional media, electronic storyboards can actively point out "breakdowns" to design teams and afford different types of "talk back" [26] and offer reflection that may not be possible with passive media [27, 138]. As described in the previous section, these "breakdowns" are critical to advancing the design process through re-evaluation of design ideas.

In the next section we will examine how electronic storyboards enable design teams to experience "breakdowns" through the physical storyboard interface like traditional media, and computationally though queries during compilation and design outputs produced by the tool.

4.3 Developing a User Study

In this section we outline a user study to evaluate the effectiveness of electronic storyboards. We present four questions to be answered by the study and outline an experimental setup involving interdisciplinary teams.

4.3.1 Study Questions

Based upon the model of design presented in Section 4.2 we describe four questions to be addressed through the user study. Questions 1 and 2 explore how electronic storyboards help design teams express and iterate on design ideas within the study. Questions 3 and 4 assess more generally the usability of electronic storyboards.

Question 1: How do electronic storyboards support design discussion?

A major drawback of existing tools identified in Chapter 2 was the inability to express designs across domains. We propose electronic storyboards as a remedy for that situation as the free-form sketching allows for expression of a large set of ideas. As a boundary object, electronic storyboards must be able to "push and pull the right kinds of information" between disciplines [141]. In the context of pervasive design, electronic storyboards must be able to facilitate discussion and expression of design properties outlined in Section 4.2.1. To evaluate this property we employ two methods: (1) four questions to assess participant sentiment regarding how electronic storyboards helped them work with each other and understand the design space from Section 4.2.1, and (2) an analysis of design conversations and activities during the study.

First, after the study each participant will be asked if: "I feel that using electronic storyboards help me...": (1) communicate with my team member, (2) understand the physical properties (size, form, placement...etc.) of the prototype, (3) understand how the user interacts with the prototype and (4) understand the computational properties of the prototype (e.g. how and when to respond to an event, when to make decisions, the overall complexity of the design... etc.). Responses to the questions are on a five point scale from Strongly Disagree to Strong Agree. This questionnaire will provide an aggregate understanding of how electronic storyboard support design discussions.

Second, while boundary objects are central to collaborative design, face to face communication and relationship building is necessary to support these objects [133, 142, 143, 144]. As Fischer described: "The interaction around a boundary object is what creates and communicates knowledge, not the object itself." [136]. To assess how these different facets support design discussions we will analyze participant conversations and actions while using each element of the electronic storyboard through audio and video transcripts of the user study. Electronic storyboards are multi-faceted boundary objects that have several points of contact. Teams can storyboard, respond to queries from the tool, and examine outputs from the tool. We we will note what types of design discussions took place (what aspects of the prototype the teams discussed) and what activities were occurring during that discussion (storyboarding, responding to tool queries, examining tool outputs...etc). Our assumption is that through conversation [145, p.371], interacting with the tool, and sketching teams explore the design space of the prototypes within the prompts.

Question 2: How do electronic storyboards support design iteration?

In time-constrained environments more iterations over time lead to better results [32]. One anticipated benefit of electronic storyboards is that they will help design teams iterate on their design process by actively promoting iteration. As discussed in Section 4.2.2 electronic storyboards are distinguished from traditional design tools by their ability to "talk back" to the design team. Through "talk back", whether computationally or otherwise, design teams can experience "break downs" that advance the design process [138]. To understand how electronic storyboards support iteration, we focus on team interactions with the queries generated by the tool, and the team's utilization of storyboard outputs to reflect upon their design. While teams may experience break downs or iteration during storyboarding, our analysis will focus on the "active" rather than the "passive" elements of electronic storyboards.

When designing boundary objects it is important that the interfaces to the object be familiar to the users [138]. As these objects are used in interdisciplinary settings, often the interface or method of interaction is defined by one particular domain. If one discipline/domain becomes dominant, it may create inequitable relationships between the collaborating parties [141]. A similar situation exists with electronic storyboards as storyboarding itself may be unfamiliar to computing participants while state charts and source code may be unfamiliar to non-computing participants. For the user study, while there is no perfect representation that can be provided, at the end of each task will we ask the participants which output representation they preferred and how they would use these representations to understand their design.

Question 3: Are electronic storyboards usable by design teams?

Beyond being a vector to understand design teams, electronic storyboards are a software tool subject to the same usability requirements as other programs. Regardless of how electronic storyboards support design discussions or iteration, if the tool is not usable by design teams then it will be of no benefit to them. We evaluate the usability of electronic storyboard through severals approaches, first we employ a System Usability Survey (SUS) [146, 147] to assess general usability of the tool. Given our Wizard-of-Oz setup, SUS questionnaires may not typically employed. Therefore our SUS results cannot be used to compare against other systems evaluated by SUS, but since it was administered to all participants in the same manner, it provides a quantitative measure of usability among the participants.

Part of the usability of electronic storyboards is that it provides benefits to the design team by facilitating discussion and generating source code that could be implement future prototypes. Code generation is anticipated to be a perceived benefit for computing participants. All participants will be asked in post-study interviews how they believed this tool could help their existing practice. Finally, with the Wizard-of-Oz setup, there may be instances where the proctor has to intervene to address unexpected situations. The causes of these situations will be noted as how they affect usability.

Question 4: How did the underlying models and transformations perform?

In a final examination we look at how the models and transformations outlined in Chapter 3 held up during the user study. When developing electronic storyboards several design choices were made regarding expressing computational information, use of natural language processing, and selecting an underlying model of computation. While these decisions were driven by our experiences and sample storyboards, user input from the study may be different. To ensure this data is collected the proctor's interaction with the tool was recorded during the study including all queries and outputs produced by the tool. Participants were not restricted in how they storyboarded, except that they use the tags, frames, and arrows to indicate information and layout. Given the variety of inputs expected, did the models described in Section 3.2.3 accurately reflect what participants produced? Additionally, were the model transformations described in Section 3.3 sufficient for the participant storyboards?

4.3.2 Experimental Setup

To assess the underlying mechanics and concepts of electronic storyboards we conducted a user study with six teams of students from various disciplinary backgrounds. In this study we used a Wizard-of-Oz setup where the teams themselves did not directly use the tool, but created typical paper storyboards that were then transfered into our electronic storyboarding tool by the study proctor. Teams would produce paper storyboards with colored annotations, as in Figure 4.3, which would be directly transfered into the electronic storyboard as in Figure 4.4. Having the proctor operate the tool allowed the design team to focus on their storyboarding and responses from the electronic storyboard,

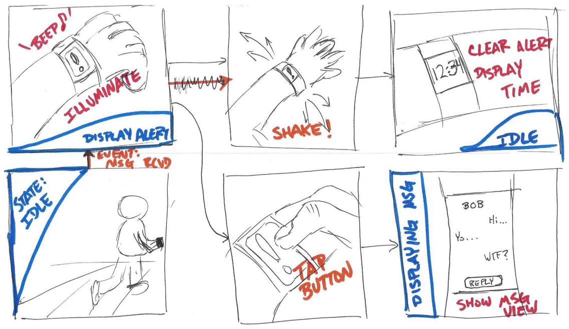


Figure 4.3: Participant Storyboard for Prompt 2

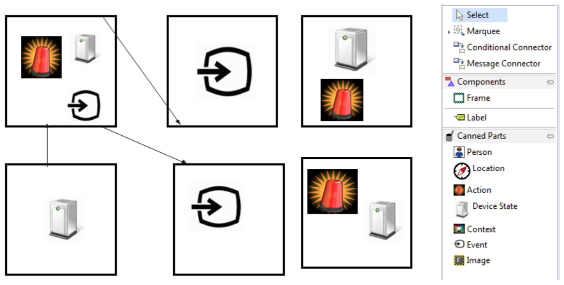


Figure 4.4: Transferring paper storyboard from Figure 4.3 into electronic format.

rather than learning how to operate the electronic storyboard itself. Conducting the user study will allow us to better understand how interdisciplinary teams use electronic storyboards and will improve future iterations of the tool.

As will be outlined in Section 4.3.1, throughout this study we observed how teams storyboarded, discussed the properties of the interactive systems, and how they responded to questions by the electronic storyboarding tool. After completing the user study each team was asked to fill out a modified System Usability Scale (SUS) questionnaire [146, 147] asking whether they agreed with certain statements about their experience. Four additional questions were added to the SUS questionnaire to evaluate whether electronic storyboards supported team communication, and understanding of computing, user-interaction, and physical form properties of storyboarded systems.

Participant Selection

Each team in the study contained two members, one had a "computing" background and the other was from an alternate discipline. Table 4.1 provides a list of study participants, their team number, and their disciplinary background. All participants were either graduate or undergraduate students at our university. The six computing participants had backgrounds in computer engineering, computer science, and human-computer interaction, while the non-computing participants backgrounds ranged between chemistry, industrial design, fine arts, and social sciences. These participants were selected as they are representative of future interdisciplinary teams that may find electronic storyboards useful.

Team	Participant	Discipline
1	P1	Computer Engineering
	P2	Fine Arts
2	P3	Computer Engineering
	P4	Chemistry
3	P5	Computer Science
	P6	Science Technology & Society
4	P7	Computer Science
	P8	Instructional Design
5	P9	Computer Science
	P10	Industrial Design
6	P11	Computer Engineering
	P12	Industrial Design

Table 4.1: Study Participants

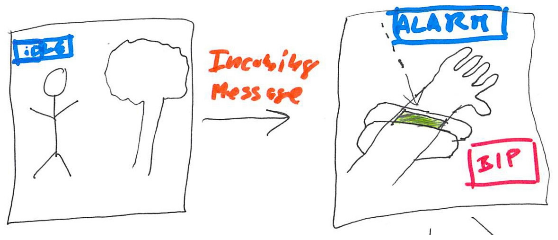
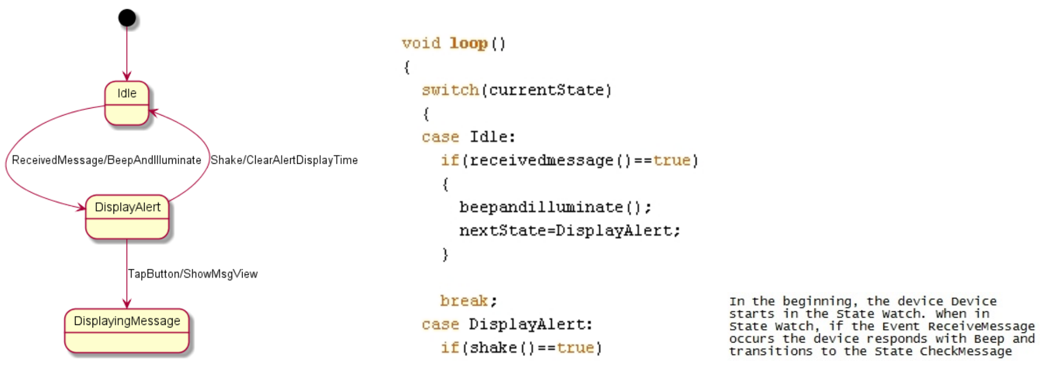


Figure 4.5: Two frames from a user annotated storyboard indicating State, Event, and Action tags

Protocol

Each team was provided 10-15 minutes of training that described storyboards and showed typical examples from industry and academia. The team was also instructed about electronic storyboards and how the team could utilize tags, natural language processing, and layout to create automatic descriptions of interactive devices. After training each team was given three prompts that described an interactive system and was asked to storyboard that narrative. The participants were not restricted in how they storyboarded, except that the narrative contained frames or arrows where necessary to determine layout. No time limit was imposed on the participants but they were naturally limited by the own schedules. Accordingly, each study took approximately 75 minutes to complete.

While storyboarding, the team was asked to apply State, Event, and Action tags from Table 3.1 to their storyboard using colored markers to indicate if that information was present in a particular frame. A sample storyboard is shown in Figure 4.5 with States, Events, and Actions indicated in Blue, Orange, and Red, respectively. These tags were to indicate the states, triggering events, and responsive actions of the storyboarded system. Once the team felt they had completed their storyboard it was implemented in the electronic storyboard tool by the study proctor. The role of the proctor was to transfer the layout and tags of a paper drawn storyboard into an electronic one as in Figure 4.3. The electronic storyboard was then compiled into the three different outputs as in Figures 4.6a-4.6c. Any questions raised by the tool (to resolve ambiguous or missing information) was relayed to the team by the proctor.


Idle
ReceivedMessage/BeepAndIlluminate
Shake/ClearAlertDisplayTime
DisplayAlert
TapButton/ShowMsgView
DisplayingMessage

```
void loop()
{
  switch(currentState)
  {
  case Idle:
    if(receivedmessage()==true)
    {
      beepandilluminate();
      nextState=DisplayAlert;
    }

    break;
  case DisplayAlert:
    if(shake()==true)
```

In the beginning, the device Device starts in the State Watch. When in State Watch, if the Event ReceiveMessage occurs the device responds with Beep and transitions to the State CheckMessage

(a) Statechart (b) Source code (c) Textual

Figure 4.6: Three representations generated for user study.

Throughout the study, participants were subject to audio and video recording. Team members were not instructed on how to work together, or what roles each member should assume. Accordingly, team conversations reported in this study are naturalistic and were not prompted by the proctor, except during the post-study interviews where participants were explicitly asked about their experience. During the study the proctor did not intervene into the team's storyboarding unless the participants were significantly off task. This occurred infrequently throughout the study, however on a few occasions teams would confuse the definitions of Event, Action, and State tags. These situations are described in more detail in the discussion section.

Task Selection

Each team was asked to storyboard three narratives that described a pervasive computing system. Prompt 1 was very simple and described a child who is wearing a smart watch. While outside playing, the watch beeps because he has received a text message from his parents. This initial prompt was expanded in Prompt 2 where after receiving the message the child could decide to press a button on his watch to display the message, or shake his wrist to make the message go away. Prompt 3 described a young girl who often forgets items when she is leaving for school. The prompt asked the team to storyboard her getting up one morning and rushing out to the bus, but as she exiting the house, the reminder system attached to her book bags warns that she forgot her science book. Following this alert, she runs back inside to retrieve her book and still makes it to the bus in time.

Prompt 1 describes baseline behavior that should be easily storyboarded by the participants. The story described in Prompt 1 is very linear and should only use a handful to tags to convey the information. An example storyboard of Prompt 1 is shown in Figure 4.5. Prompt 2 is more complex and requires the participants to apply more tags, but also use arrows within the storyboard to indicate branching behavior as the user has to choose between reading or dismissing the message. Prompt 3 presented a different challenge as the story was more complex and potentially required the participants to use loops for entering and exiting the house with and without the book. Furthermore, the means by which the reminder system knew the book was missing was not specified and provided the teams an opportunity to discuss what technologies or sensing systems might be used.

The prompts selected for this study do not fully explore the design space as outlined in Section 4.2.1. The three prompts only cover a some set of possibilities that relate the computing, physical, and user interaction elements of pervasive systems. It is not our intention to explore the entire potential space, but to provide prompts that would be reasonable for design teams during the early phases of prototyping. Consequently, these prompts strike a balance between explicitly defining the device behavior and allowing the design teams to improvise and be creative. As we will see in the next chapter several teams engaged beyond the prompt as written.

4.3.3 Assumptions and Limitations

In conducting this study there are several underlying assumptions and limitations that must be examined. First, we assume that participants have not previously worked together. Electronic storyboards were derived from our experiences with novice student design teams. Their difficulties of coordination and expressing concerns across disciplines may be more pronounced compared to teams that frequently collaborate or have prior interdisciplinary experience. Thus the amount of time spent on particular tasks or the content of design discussions may be different with more experienced teams.

Second, the types of prompts provided to the teams directly impacts how challenging their tasks may be and the properties of the system that are discussed. Prompts 1 and 2 were designed to test the feasibility of electronics storyboards - if teams cannot express prompts of this complexity it is unlikely that the tool is usable. Prompt 3 was more open-ended to allow teams to discuss various implementation approaches and technologies. In all of the prompts, no property (as defined in Section 4.2.1) was under heavy constraint. Teams did not have to optimize their design to meet stringent computational requirements nor were the prompts so complex to make the user interaction cumbersome. We do not anticipate electronic storyboards can handle all potential pervasive computing use cases. Future testing is required to see at what point electronic storyboards "break" for a given prompt.

Finally, our research methods are limited in scope to the general "topic of conversation" and do not provide finer grain results about the team's conversations. It is beyond the scope of this work to analyze which properties of the system each disciplinary group discussed (e.g. Did those with computing background talk about *computing* properties more?). For the same reason, information about sub-tasks (e.g. Who did most of the storyboarding/tagging?) is not considered in this work. This information would be useful in future studies to understand in more detail what types of information each discipline communicates, finds important, and which types they have difficulty reconciling with their own practice.

4.4 Summary

In this chapter we have proposed a model to evaluate electronic storyboards as a boundary object. Recognizing that electronic storyboards are a complex socio-technological system, we proposed qualitative and quantitative methods for their analysis. Four user study questions were outlined that assess how electronic storyboards support interdisciplinary teams in expressing design ideas, reflecting on those ideas, determine the usability of electronic storyboards, and evaluate the reliability of its underlying models and transformations. We examine the results of a user study in the next chapter that answers these questions.

Chapter 5

Results

This chapter presents the results of the user study outlined in Chapter 4. We address Research Question 3 by answering four questions major questions regarding electronic storyboards. First, in Section 5.1 we examine how storyboarding and tagging support discussion of pervasive computing design properties. Second, in Section 5.2 we show how tool queries and outputs support iteration and revision of design ideas. The remaining Sections 5.3 and 5.4 deal with the usability of electronic storyboards and how the underlying model-driven engineering approach performed during the study. Finally, we summarize these results and evaluate our design choices throughout this dissertation.

5.1 Supporting Design Discussions

This section provides qualitative and quantitative analysis to explore how electronic storyboards support design discussions about pervasive computing systems. During the study we observed that teams discussed design properties via two main mechanisms: the manual storyboarding and depiction of the prompt and the application of tags to the storyboard. Overall, storyboarding and tagging facilitated significant discussion of the computational properties of the device and moderate discussion of the user-interaction. Very infrequently did the design teams discuss the physical properties of the device.

5.1.1 Discipline-based Sentiment

After the user study participants were asked to respond on a five-point scale whether, "I feel that using electronic storyboards helps me...": (1) communicate with my team member, (2) understand the physical properties (size, form, placement...etc.) of the prototype, (3) understand how the user interacts with the prototype and (4) understand the computational properties of the prototype (e.g. how and when to respond to an event, when to make decisions, the

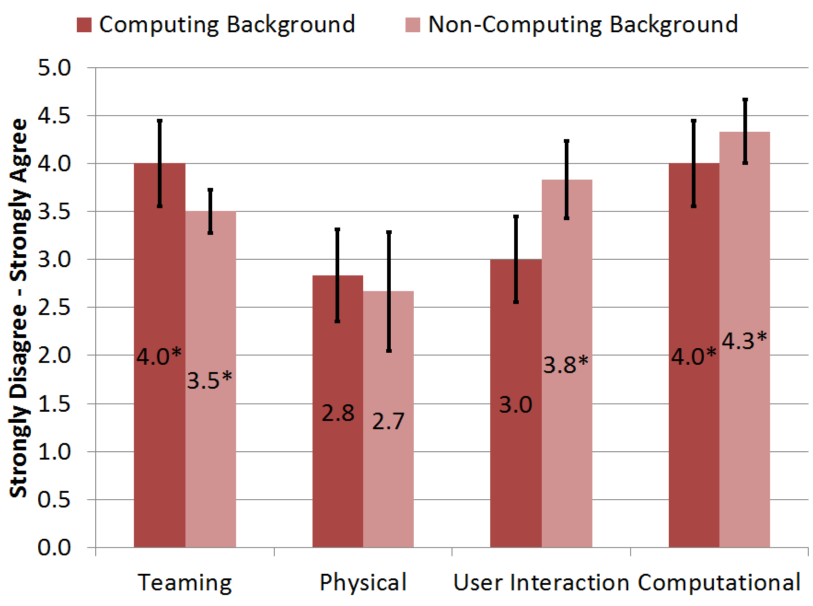


Figure 5.1: Discipline-based Sentiment

overall complexity of the design... etc.). Participants responded on a 1-5 scale whether they Strongly Disagreed (1), Disagreed (2), Neutral (3), Agreed (4), or Strong Agreed (5) with a statement.

A one-sample t-test was used to compare the mean participant responses against an assumed "neutral" response. A null hypothesis ($H_0 : \mu = 3$) assumed that participants were "neutral" towards a question, while the alternative hypothesis ($H_1 : \mu > 3$) assumed they "agreed" with a question and found that aspect helpful. Hypothesis tests were conducted at 95% confidence. Figure 5.1 shows the mean score for computing and non-computing participants with one standard error in each direction. Asterisks on the scores indicate statistically significant agreement with the question. Results show that computing (p=0.036) and non-computing (p=0.037) participants agreed that the electronic storyboards helped them communicate better with their team members, computing (p=0.037) and non-computing (p=0.0052), agreed it helped them understand the computational properties of their prototype systems. Only non-computing participants (p=0.046) felt electronic storyboards helped them better understand the user's interaction with the prototype.

The quantitative results presented in this section provide backing to the qualitative results in the remainder of this section. We will see in Sections 5.1.2 and 5.1.3 support discussion of computational properties and user interaction. As foreshadowed in Figure 5.1, few teams discussed the physical properties of their storyboarded systems.

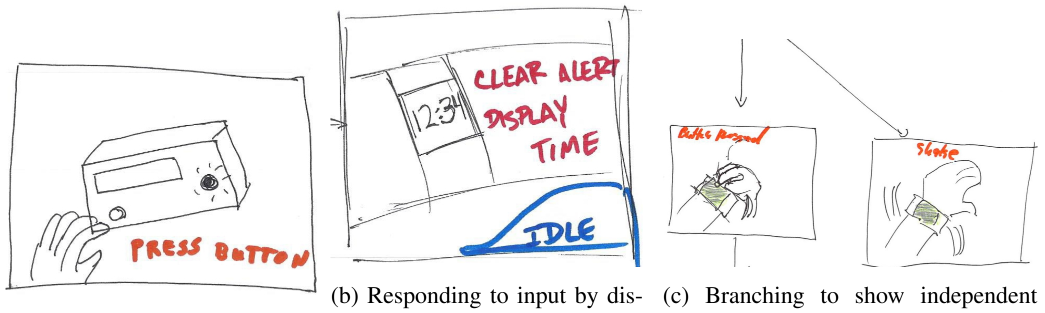
5.1.2 Storyboarding

Storyboarding the study prompts required the teams to represent the device, how it received input, and how it responded to certain situations. While drawing the storyboard, depicting simple responses such as "beeping" or "pushing a button” and more complex interactions such as how to represent an “incoming message”, provided teams a way to discuss how information should be expressed and qualified. Example storyboards are shown in Figures 5.2a and 5.2b. “*P2: to represent a message coming I would draw an arrow with a dash to say that some information is coming. Through a satellite.*” Drawing these interactions also facilitated discussion about the specifics of how an event should occur. For example, when discussing the alert sound of the watch: “*P2: the beep has to be high-pitched so it can be heard in any kind of environment.*”. Team 1 expanded this discussion to consider the situation of deaf persons and whether or not the current system was appropriate, “*P2: I don’t know if it’s part of my job here but imagine somebody who is deaf. It could vibrate.*”

While Prompts 1 & 2 largely described the technology in the system, Prompt 3 did not specify how the reminder system knew a book was missing and provided teams an opportunity to discuss what technology or sensing methods were required. “*P10: the point where it’s sensing she’s missing something, do we just need to make up the technology there? P9: we can say, we can make it up. I think the input would be.. we can suggest some sensor.*” Team 1 discussed a specific technology and whether the device would scan continuously or only at certain time, “*P2: somehow I don’t know how it’s supposed to work. I guess you’re supposed to have some barcode or something, or some cheap? P1: or RFID or something like that. But that’s the question, is it constantly scanning or is it a one-time scan, and then it’s done for the rest of the day.*”

When drawing the storyboard the teams had to lay out the set of frames that described their story. During training the teams were told that branching or divergent behaviors could be expressed using arrows to connect independent frames. Prompt 1 was generally described with a linear flow while Prompt 2 had branches to show divergent behavior. ‘‘*P10: so one branch needs to be shaking to make it go away the other needs to be tapping.*”. An example of branching is show in Figure 5.2c.

While we expected the storyboards to facilitate discussion about the physical form of the device, this was rarely discussed by the team. This finding follows from the questionnaire results in Figure 5.1 where physicality was ranked the lowest. One explanation maybe that while the prompts did not specify how the devices should look, they used canonical devices that have an implied or known form. For example, two prompts dealt with a smart watch and the other a reminder system that attached to a book bag. Given the existence of consumer wearable products, there was not significant discussion of the watch’s form and many took on a squarish shape with one team commenting, “*P12: I don’t know why I resorted to a square, I guess a square made me think of a smart watch.*” Similarly, the reminder systems was described as something to fit on a book bag and had limited interaction with the user (a single button press). As the form of this system did not impact the functionality or the user interaction it was unlikely to warrant significant discussion.



(a) The user pressing a button. (b) Responding to input by displaying to the screen. (c) Branching to show independent paths

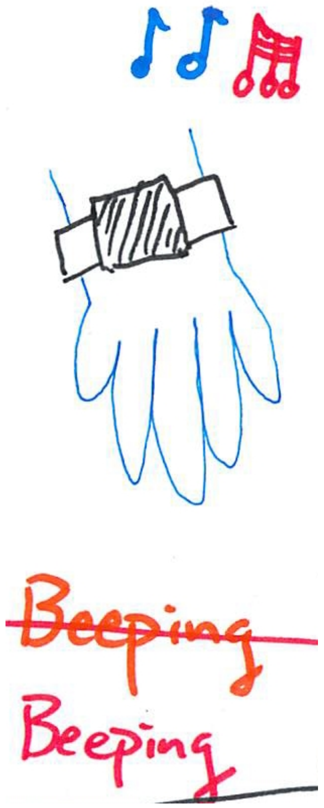
Figure 5.2: Use of storyboarding to user interaction, logical flow and responsive actions of the device.

5.1.3 Tagging

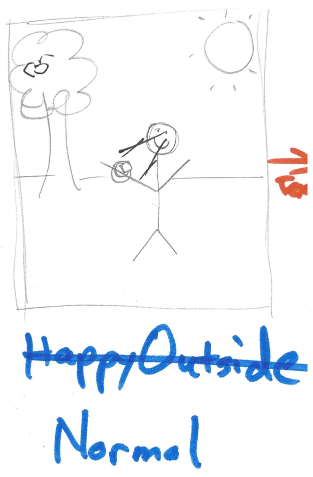
Using the State, Event, and Action tags, the design teams annotated their storyboards to indicate important information regarding the prompt. These tags directly support discussion of computational design properties as the team must determine what information to tag in the storyboard, what to name tags, and which tags to apply. Often this involved discussion of what information a device must need before taking some action. A few examples of this discussion come from the reminder system prompt where the device needs to check for a missing book once the user exits the house. "*P5: (remember) the box needs to be in some state that triggers it to check for the text book. P6: departing is fine. P5: or ah, missing item?* " and "*P7: It's got to be in a state where it's like, I need to do an inventory.*" Another group discussing the same prompt commented, "*P3: so it doesn't know about that, but it does know leaving house again. But it does nothing because there's some sort of trigger on having a book. P4: right, because now it's satisfied.*" In determining the name of the tag, the team is deciding what the device is actually doing: "*P1: so do we want to call this beeping, processing.. P2: or alarm?*"

However, the tags were not fool proof and the difference between events and actions were often confused with one being substituted for the other. The explanation provided to teams during the study was that Events are triggers or inputs to the device, and that Actions are outputs or responses of the device. Frequently participants would interchange the two terms and confuse Events for Actions. This confusion normally occurred during the first or second prompt and the participants would eventually corrected themselves. Figure 5.3a shows Team 2 mis-tagging the events and actions of the smart watch. Soon after drawing "beeping" they realize their error, " *P3: event is orange? P4: event, it beeps. and action is red, read message. P3: or it's the other way around, the event is an input to the device (re-reading the study instructions). The event is actually message received and the action is beeping. P4: oh I see.*"

Similar difficulties were found with State tags as teams considered "state" to be the state of the person in the storyboard rather than the state of the device. Team 6 confused State tags as shown in Figure 5.3b. "*P11: so here, happy outside is describing the state of the frame, as opposed to the state of the device. So this would be Normal? P12: on wrist? P11: the normal state. This would be P12: oh that makes sense. It beeps, it's the device.*"



(a) Event and action tags.



(b) State tags.

Figure 5.3: Mis-tagging storyboarding information.

One explanation of tag confusion could be that participants focused more on the *person* in the storyboard rather than the *device*. Considering that teams were provided a narrative that described a human user and that person's interaction with some electronic system, it might be more natural for the participants to focus on the human rather than the computer. Given our definitions of event and action, what could be an action to the user, such as pressing a button, is equally described as an event to the device, receiving the button press. P6 commented in the post-study interview, "*P6: In the first prompt the human wasn't doing anything that wasn't directly connected with the device. In the second one she was going in and out of the house, she was looking at the bus and stuff like that. It contributed to this confusion I think when we drew the story first and then had to tag it afterwards.*" We discuss the impact of this "user-centered focus" in Section 6.1 where we present guidelines for future design tools.

5.2 Supporting Design Iteration

Once the team had drawn and tagged their storyboard, an electronic version was created by the proctor and entered into the tool. While compiling this storyboard into a timed automaton the tool would ask questions to resolve ambiguous or missing information. Examples of this situation included asking for missing events or actions between states or

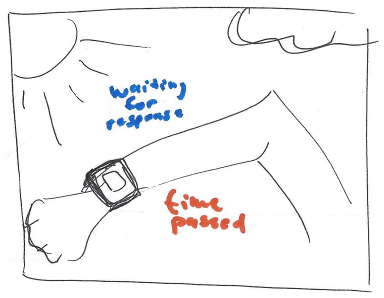


Figure 5.4: A tagged event with no corresponding action identified by the tool.

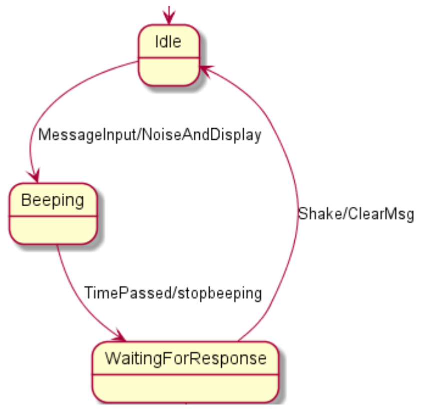


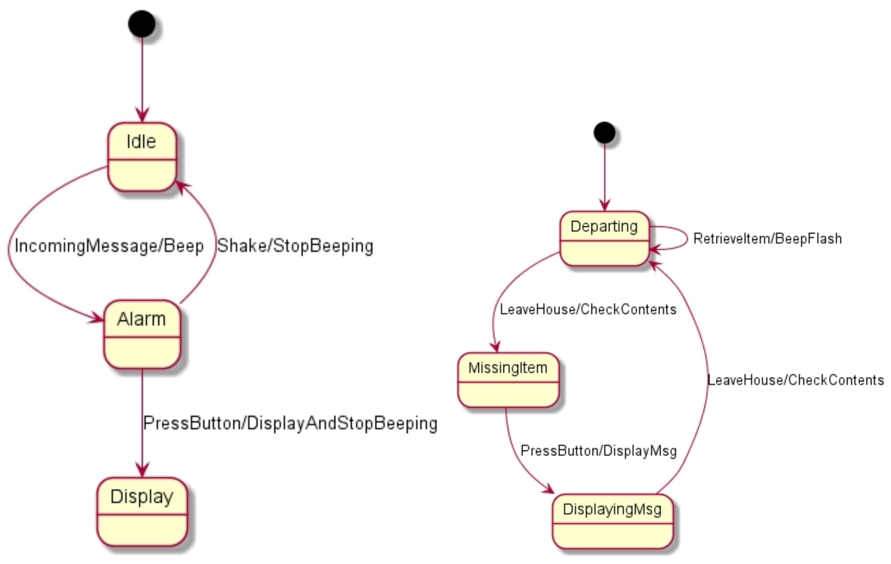
Figure 5.5: Portion of a generated state chart with discovered action "stopbeeping".

resolving the relationship between events and actions if multiple were present. This questioning benefited the design teams by providing new representations of their design through which they can evaluate their depicted behaviors. Additionally, this process can "check" the storyboards to determine if triggering events have been correctly paired with responsive actions, or if extraneous tags are present.

5.2.1 Iteration through Queries

Queries from the tool often helped teams find missing information from their storyboard. An example of a missing action found by the tool is shown in Figure 5.4. Team 5 wanted the smart watch to stop beeping after some time and tagged the event "time passed". However, the team did not tag any related actions and when parsing the storyboard the tool could not pair "time passed" with an action. Seeking to find a responsive action the tool asked the team to specify whether or not there was an associated action. This query was displayed on the tool console and relayed to the design team via the proctor, "*Proctor: what is the response to the event time passed? P9: stop beeping? P10: yeah silenced.*" The proctor then typed in "stop beeping" into the tool console to provide the team's response.

Figure 5.5 shows the resulting state chart where the action "stop beeping" has been added by the tool. Similarly, Team 6 tagged an action called "pack bag" in their reminder box storyboard. This action was not a required part of the storyboard and was an extraneous tag. The tool tried to pair that action with multiple events throughout the storyboard.



(a) Deadlock (b) Missing item.

Figure 5.6: Using statecharts to evaluate design ideas.

Each time the design team declined the tool's choice. By querying the design team the tool was able to determine that "pack bag" was not important to the operation of the storyboarded device and eventually dropped the action.

5.2.2 Iteration through Code, Text, and State Charts

In addition to the questions posed by the tool, providing a new output representation of their storyboards allowed the teams another view of their storyboards to discover missing information. In particular, one team discovered that they had not specified a certain transition and also that the storyboard they had drawn was stuck in a loop. Viewing the statechart in Figure 5.6a their storyboard the team commented: "*P1: You see how we have this display that's hanging off, how do you stop it from displaying?*" In discussing the problem, the team debates different ways to alleviate the situation "*P2: I think it should be part of a condition, not the result of an event, you know it could stop displaying automatically. P1: it could. P2: or it could respond to an action, an event, press again. P1: or could it be shake?*" Similarly, Team 3 used the statechart in Figure 5.6b commenting, "*P3: I think that there needs to be a connection between, ... something is missing between displaying the message and retrieving the item. She can't go straight from ... she needs to retrieve the item before she leaves the house.*"

Across the teams, most members preferred the state chart view as a primary source, with the text and source code being secondary. "*P4: The flow diagram [state chart] makes it very simple. The words are kind of nice if I had a question. The code, I don't know what it means.*" Several participants commented that the text view was helpful, but it became cumbersome as the complexity of the design increased. "*P6: And the text. It was sort of a natural thing that I went to the first time, and now that it's long I think that the visual portion is much easier to understand and faster.*"

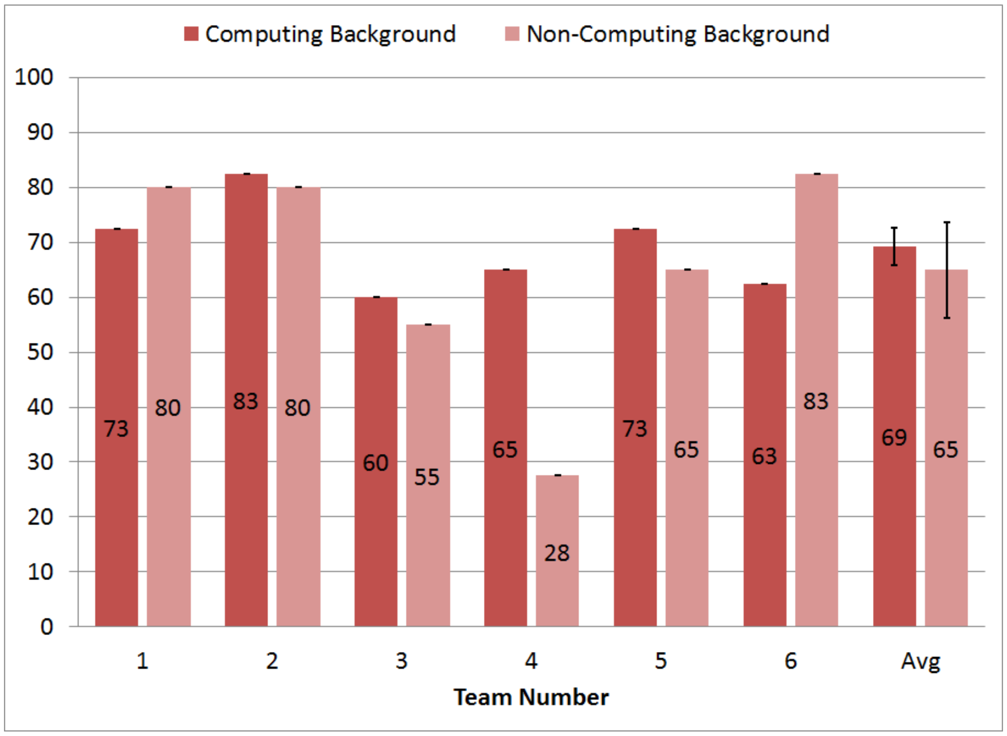


Figure 5.7: System Usability Score

Even team members who had not been accustomed to state charts found them more useful as the study went on. "*P6: I like the chart view better now. P5: it's definitely the easiest one for me to parse.*" Similarly, some participants found the source code would be helpful during an actual implementation, but was not helpful when understanding the high-level behaviors of the device.

5.3 Addressing Usability

In this section we address the general usability of electronic storyboards. In Section 5.3.1 we discuss the results from the SUS questionnaire administered after the study and examine how long participants took to accomplish the study. In Section 5.3.2 we report on participants comments on their perceived utility of electronic storyboards and how it might help their practice. Finally, in Section 5.3.3, we examine points in the study where the proctor intervened when groups were off task.

5.3.1 SUS and Task Times

System Usability Scale

This section presents the results from the the post-study usability survey that was administered to each team. Figure 5.7 shows the individual usability scores from the six teams in the study along with the average for each disciplinary background. In these results we differentiate between our computing and non-computing participants. The range of usability scores indicates that participants had varying positive and negative experiences with the study. While half the participants provided scores over 70, which indicates an 'average' result, one participant in Team 5 had significant difficulty and reported a score of 27.5. Looking at this participant's individual responses, they were not very confident using the system and felt it was difficult to use.

Overall, participants with a computing background provided a higher usability score. This outcome may be expected as during the study the computing participants would often guess or infer "what's under the hood" of the electronic storyboarding tool. Frequently they would anticipate future problems or even indicate behaviors the tool "should handle." This extra knowledge may have made them more confident in using the tool. In one session the team drew a storyboard that contained looping behavior and they discussed if that was acceptable. The computing member commented, "*P3:...[the tool] should be able to handle it.*"

Task Time Based upon Prompts

All teams were able to complete their assigned tasks during the study and agreed that the outputs produced by the tool described their system. Figure 5.8 shows task time for each group during the study. The reported time (in minutes) indicates how long it took the teams to arrive at a final storyboard that they agreed implemented the prompt. The submitted storyboard may have been their first draft, or one that had been compiled and modified several times. Examining the task times, Prompt 1 and 2 took less than 15 minutes each. State chart solutions from Prompt 1 were generally "linear" as in Figure 5.9a. Prompt 2 was an extension of Prompt 1 and asked teams to enable the user to dismiss or read the message that was received. Figure 5.9b shows a typical solution for this Prompt. Not all solutions to Prompt 2 where exactly as Figure 5.9b - some teams added additional features for the watch to return to an idle state by shaking the wrist or by ignoring the message. Overall, the results of Prompts 1 and 2 indicate that teams can use electronic storyboards at a basic level by indicating branching behavior within the storyboard.

Prompt 3 was more difficult for the participants and took between 13 and 29 minutes across all teams to complete. Prompt 3 was more ambiguous about how the device should be implemented and caused the team to discuss various solutions. For example, the teams were not told how the reminder system knew an item was missing and it did not specify what happened after the user returned inside and retrieved the book. Consequently the storyboarded solutions varied across participants. Figure 5.10 shows two different solutions to Prompt 3. One team implemented

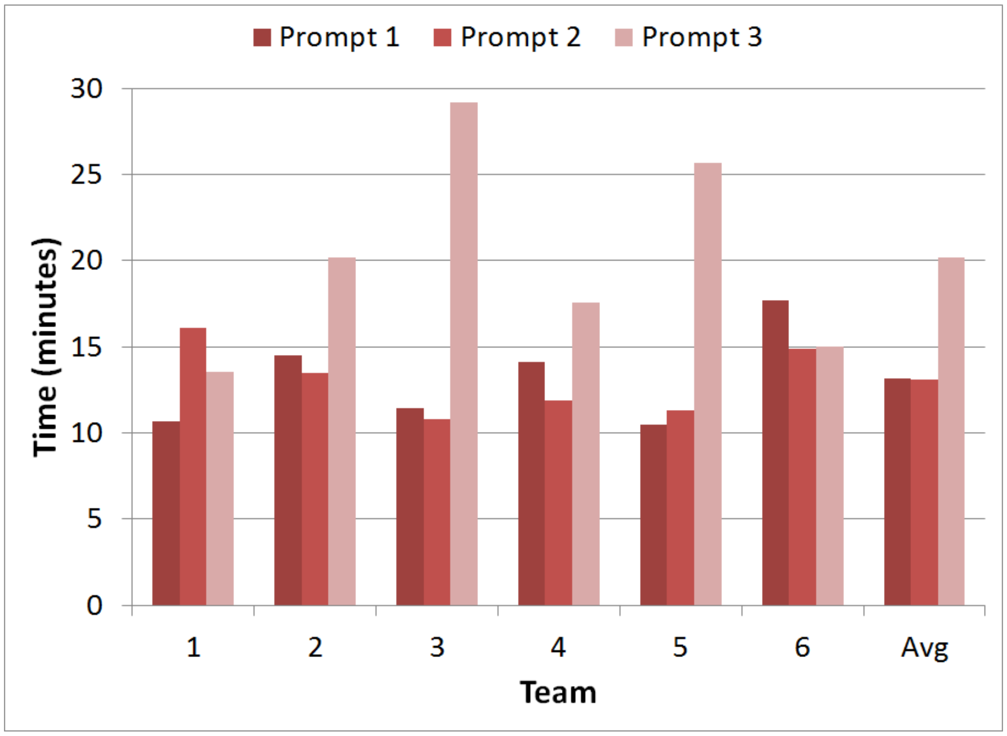


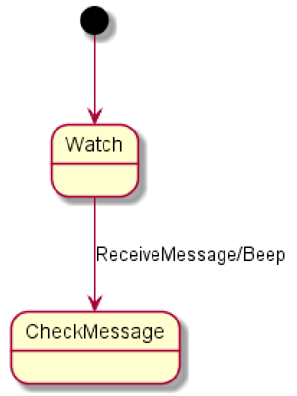
Figure 5.8: Time until final storyboard submission for each group.

the system exactly as described in the prompt as in Figure 5.10a, while other teams tried to express looping behavior as in Figure 5.10b. The main concern for teams was how to express conditional events upon exiting the house ,e.g., to sound an alarm when exiting the house without the item and not to alarm if the item was present. While teams could verbally express what they wanted the system to do, some found it difficult to sketch that behavior within the storyboard.

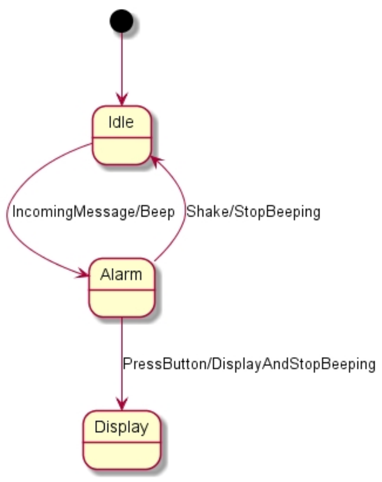
One reason for the difficulty with Prompt 3 was the lack of variables or complex conditional events within the storyboard. Conditional events, such as "A *and* not B" are supported by the underlying models in electronic storyboards but were not exposed to the users to avoid confusion. We anticipated these complex events would be given a event name, and the complexity of that event would be address during implementation. We address the relationship between the tool interface and its underlying models in more detail in Section 6.1.

Task Time Based upon Activities

Examining the team activities in more detail, we divided the team's activities into three phases: storyboarding and tagging, responding to tool queries, and examining outputs produce by the tool. Figure 5.11 shows the total amount of time each team spent on each activity during the study. On average, teams spent 33 minutes storyboarding, 4 minutes responding to queries, and 8 minutes examining the tool outputs. These times are the aggregate of team activities across all prompts - teams did not spend 33 continuous minutes storyboarding. As discussed in Section 5.2,



(a) "Linear" solution for Prompt 1



(b) Branching solution in Prompt 2

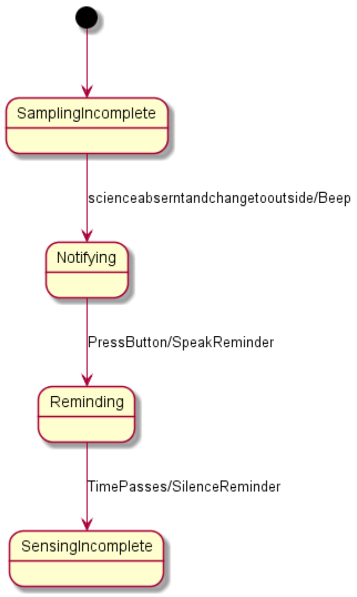
Figure 5.9: Typical Prompt 1 and 2 solutions

some teams iterated on their ideas based upon tool queries or examining the tool outputs. Figure 5.11 indicates the importance of storyboarding and sketching as it was the primary activity that teams engaged in. Ideally, the time spent responding to queries and examining outputs would be minimal, indicating that teams "correctly" storyboarded their systems and the tool produced outputs that reflected the team's ideas.

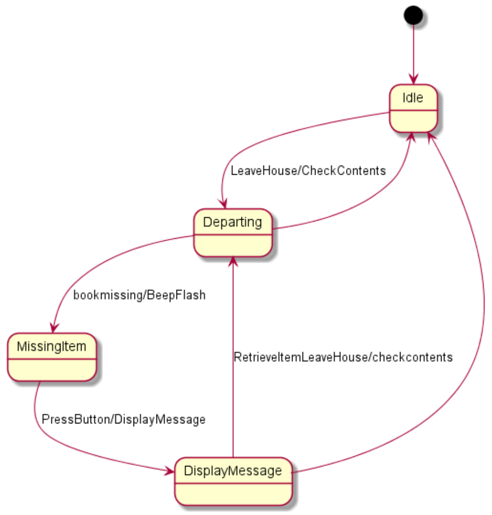
5.3.2 Reported Tool Benefits

When developing electronic storyboards we anticipated that computing participants would find automated code generation beneficial. From the post-study interviews, four of the six computing participants expressed a negative or doubtful regarding the benefits of code generation. P1 commented that he would rather code the device outright. P5 expressed skepticism that code generation could produce bug free code based upon his experiences with the current tool, "*P5: It's most useful for like template to do something and I go fill in the specifics. I would be, based on what we just discovered, there's a lurking bug that I might hit or not hit. I would find the code generation useful that I was absolutely confident it didn't introduce a bug.*" P9 felt that code generation could be useful for novices as a means to find a direct translation from state charts to source code. The two other computer participants expressed support for code generation with one commenting specifically that it would be helpful to generate skeleton code or templates and then modify the results.

In contrast to these opinions, P10 who was an industrial designer, commented positively on code generation and related it to her existing practice stating, "P10: *yeah definitely, especially in the early stages I would only say that because we would like make storyboards in [Adobe] Illustrator to make them pretty. So obviously this (generated code) isn't a pretty version that we would consider, but in the earlier stages I would say yes conceptually.*"


SamplingIncomplete
scienceabserntandchangetooutside/Beep
Notifying
PressButton/SpeakReminder
Reminding
TimePasses/SilenceReminder
SensingIncomplete

(a) Prompt 3 Linear



(b) Prompt 3 Looping

Figure 5.10: Varying solutions for Prompt 3

Two reasons may help explain the participants negative response to source code generation. First, the tool used in the study is experimental and the participants had never worked with it before. Perhaps the beta nature of the tool and their limited exposure to it contributed to their unease. Second, the computing participants all had significant programming experience and were graduate level students in either computer engineering or computer science. Given their level of expertise they may feel more confident relying on their own skills.

5.3.3 Proctor Intervention

One a handful of occasions the proctor intervened into the user study to correct some participant actions. These situations occurred when the team was outside the parameters of the study or had directly requested help from the proctor. Minor interventions by the proctor included asking the participants to clarify the spelling of a tag while entering the storyboard into the tool or asking the teams to number their frames to clarify layout. More extensive interventions occurred when a team had deviated from the establish protocol such that it would impact the study results. This happened on two occasions when the proctor reminded teams how tags were defined. As discussed in Section 5.1.3 tagging was frequently used by the teams but at times they confused events for actions or labeled states as the states of the person in the storyboard, rather than the state of the device. A final intervention occurred with Team 6 due to a concern that they were "over-tagging" states within their storyboard and would make the final compilation challenging for them. The team was assigning a state tag for every frame which was not required. The team was notified of this but continued with their storyboard and completed the task.

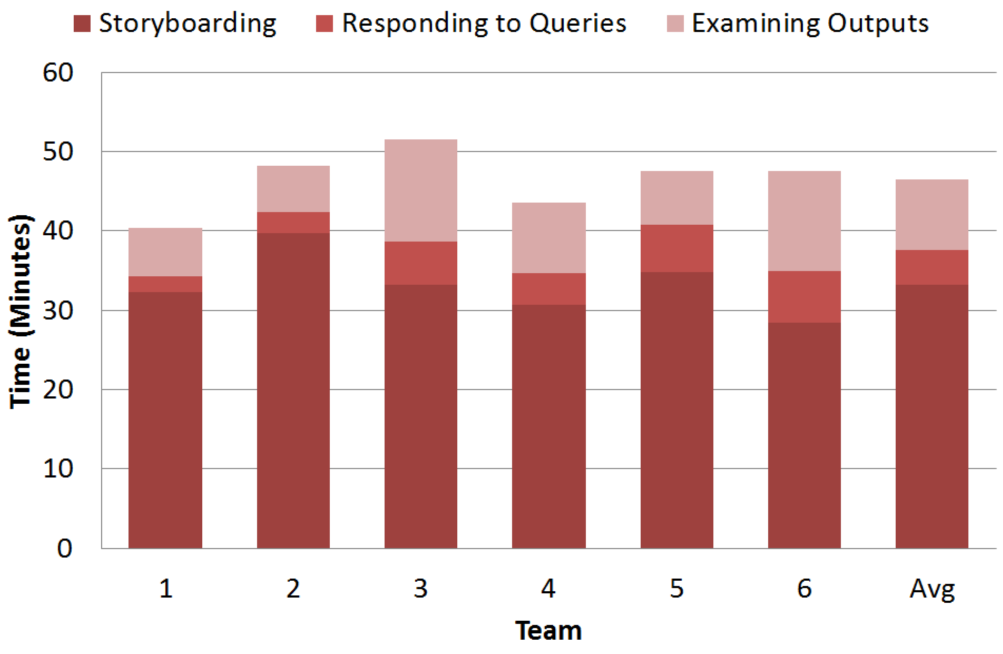


Figure 5.11: Team Activities by Time

5.4 Assessing Models and Transformations

The previous questions have analyzed team performance and usability of electronic storyboard. In this section we look at how the underlying models from Chapter 3 performed during the study. Notably we examine how our assumptions in layout and tagging were tested.

First, no storyboards generated by the teams were outside the model parameters. All teams produced storyboards with frames, connectors, and tags that could be directly entered into the electronic storyboard tool. However, one unexpected condition was found early in the study and corrected. Team 1 frequently used loops to describe their prototype's behavior. Their use of loops made it difficult to find a "starter frame". A starter frame is considered the first frame of the storyboard and where Layout Analysis, described in Section 3.3.1, begins. The original definition of a starter frame was the left-most frame with no inbound connections or neighbors. Team 1 had loops back to their starter frame and required the modification to query the proctor to manually select which frame to begin layout analysis.

Related to layout, an error was found in local synthesis where three blocks of frames joined by connectors. Usually a team would have one large block of frames describing the main story, and then one or two branches off the storyboard. However with three blocks of frames linked $A \rightarrow B \rightarrow C$, the recursion in local synthesis did not reach block C. As the three blocks were actually linear, these were not branching conditions, blocks A and B were joined to implement synthesis $AB \rightarrow C$.

Finally, given the large number of tags used by participants all potential query types were encountered during the study. Participants encountered queries where: (1) all information was specified and the tool was confirming with the user, (2) missing actions or events were to be manually specified by the users, and (3) multiple events and actions were found and had to be resolved through iterative questions. This latter type of queries, where multiple events and actions were present, were most difficult for teams. Teams never disagreed with confirmation of the type-1 queries and frequently used type-2 queries to iterate on their designs or discover missing information as in Section 5.2.1.

5.5 Summary

This chapter presented the results of a user study to evaluate the impact of electronic storyboards on interdisciplinary teams. With these results we now reconsider Research Question #3 which stated: *What is the impact of an electronic storyboarding tool on interdisciplinary teams during prototyping?* Overall there are five main findings:

Finding #1: electronic storyboards support discussion of design properties through physical storyboarding and through tagging of events, actions, and states within the storyboard. These discussions help teams explore how the device interacts with the user and the logical progression of the device to meet the user's expectations. Some teams found "virtual" items like "receiving a message" or "scanning" difficult to visually represent.

Finding #2: electronic storyboards support iteration on design ideas through queries and evaluation of tool outputs. Queries and outputs provided an additional "representation" of the storyboarded systems for teams to consider and helped them identify missing information in the storyboard and resolve the exact relationship between events and actions tagged in the storyboard. Tagging was more difficult for some teams as they focused on the user rather than the device.

Finding #3: electronic storyboards are usable by interdisciplinary teams. All teams within the study were able to utilize sketching, framing, and tagging to express their design ideas. All teams were able to complete their assigned tasks, respond to tool queries, and understand the tool outputs. Computing participants were mixed on the anticipated benefits of code generation while some non-computing participants responded positively.

Finding #4: natural language processing was used less frequently than expected during the user study. While many design teams had textual annotations within their storyboard teams placed many tags in their storyboards. During local synthesis the multitude of tags was sufficient to resolve any ambiguous information and tool tool queries did not present any results from natural language processing.

Finding #5: participant response to generated code and tool outputs was more mixed than expected. Computing participants reported positive and negative attitudes towards generated code. However, all participants were able to use state charts to examine the storyboarded systems.

Finding #6: electronic storyboards may be more useful for interdisciplinary coordination rather than rapid prototyping. One anticipated benefit of electronic storyboards was the ability to derive code from the storyboards. However, in Section 5.3.2 we found computing participants did not see this benefit. Returning to the challenges of interdisciplinary design described in Section 2.2, electronic storyboards performed well by enabling participants to express design ideas, concerns, and iterate on their work. Establishing what some system "should" do is at times more difficult than implementing that specific behavior. Future code generation can be made better but only if the starting specification defines the "correct" system. The user study has shown that electronic storyboards enable teams to quickly arrive at that specification across disciplinary boundaries.

In the next, and final chapter, we review our work and look towards future directions in electronic storyboarding and it's application to additional domains.

Chapter 6

Conclusions and Future Work

Motivated by the needs of interdisciplinary design teams this dissertation has explored the development of a novel electronic storyboarding tool and its impact on interdisciplinary teams. Following a model-driven approach, a tool was developed to utilize tagging, natural language processing, and layout analysis to transform ambiguous storyboard information into models of computation based upon timed automata. Modeling the activities of design teams using boundary objects a user study was conducted to determine the impact of electronic storyboarding on teams. Results from the study indicate that electronic storyboards: (1) support discussion of design ideas through storyboarding and tagging, (2) support design iteration and reflection based upon tool queries and tool outputs, and (3) were found usable by participants.

In the remainder of this section we reflect on our work and return to Research Question 3 to offer design guidelines for future tools based upon electronic storyboarding principles of sketching, tagging, and active interpretation. Additionally, we points towards future applications for electronic storyboards in architecture and fashion design.

6.1 Reflections and Guidelines for Future Design Tools

In this section we reflect on our study and present guidelines for future design tools based upon the core ideas of electronic storyboard - namely inferring design artifacts from informal sketches and storyboards. In Section 6.2 we describe a few potential future applications.

One important guideline for future interdisciplinary design tools is to *retain free-form sketching*. As Buxton noted, “any place that I have seen design... it has been accompanied by sketching.” [1, p.97]. This statement is echoed in our results in Figure 5.11 where participant teams spent the majority of their time sketching. Sketching within design tools provides the design team a wide lexicon of symbols and ideas to drawn from. In contrast to icon-based tools (like dTools [36] or GALLAG Strip [148]) sketching allows teams to express their ideas in whatever form they can

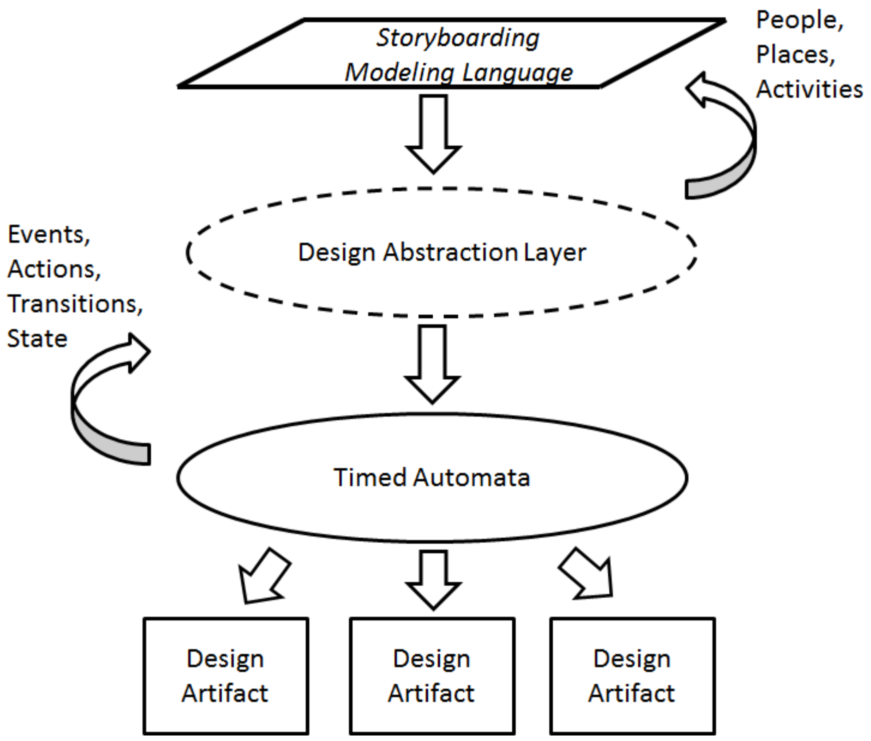

Storyboarding Modeling Language
People, Places, Activities
Design Abstraction Layer
Events, Actions, Transitions, State
Timed Automata
Design Artifact
Design Artifact
Design Artifact

Figure 6.1: New model to isolate users from underlying model of computation

draw and not in the symbols provided by a tool. This enables a sketch-based tool to target a larger set of applications without extension to the underlying language. However, this sketching comes at a price, as an arbitrary sketch cannot be interpreted by a computer without significant image recognition and training, which would defeat the point of free-form sketching. Our compromise on this issue was to use tagging that was understood by the interpreting system. While the tagging system was usable by teams it caused difficulties at times based upon the number of tags present in the storyboard. We discuss this issue further in our next guideline.

Another guideline for future design tools is to *isolate the end-user from the model of computation*. In electronic storyboards, the choice of tags in Table 3.1 was motivated by the need to describe pervasive computing systems at a high-level, but also facilitate easy mapping between storyboards and the underlying timed automaton as in Figure 3.5. As reported in Section 5.1.3 teams had difficulty at times with tagging because they focused on the user in the storyboard rather than the device itself. Additionally, when teams had difficulties with queries it was likely because they had tagged "too many" events and actions and were having difficulty selecting between their choices. Part of the problem was the inadequate responses provided by the tool, but another part is that the timed automaton model was "too close" to the users and exposed details that they may not find easy to consider. One indicator of this "closeness" was the need to tag States within the storyboard. We could find no way to map from storyboards to timed automaton without some notion of state within the storyboard. While Events and Actions could be found with natural language processing, as discussed in Section 3.2.2, we could find no analogue for State information.

One solution to this problem could be to choose another model of computation. However, depending on the model selected the same issues of "closeness" may arise. We propose in Figure 6.1 that an additional modeling layer be inserted to isolate the user from whatever model of computing is selected. This new layer could be more "user focused" based upon the observations in Section 5.1.3 that showed some participant focused on the user rather than the underlying device. This isolation may become more important based upon the role of computing participants in future tools. As described in Section 5.3.1 some computing participants determined what was going on "under the hood", however some participants (see P8 in Figure 5.7) had difficulty with the device-based tagging.

Finally, *focus on the front and middle* of the tool. By this we means to spend significant development time on the user interface and the underlying models of any future tool. While we expected that the "back-end" of the tool, such as generated source code, textual descriptions, and state charts, would be attractive benefits for the participants, we observed that they major benefit was providing teams an accessible interface for design ideas and an ability for them to reflect on those ideas. In our conversations with design professionals, active feedback and critique of design ideas has been a compelling feature. The benefits derived from tool outputs can be very application specific. Generating multi-threaded code, as in Section 3.3, may have little benefit for simple single device applications. Whereas automatically understanding context and interfacing with the Context Toolkit [117] would only benefit contextual applications. Overall, the outputs of the tool can be made arbitrarily useful depending on the target application. However, the ability to express high-level ideas, and receive automatic feedback from those ideas, is cross-cutting and benefits a wider variety of applications.

6.2 Future Work

Our study of electronic storyboards focused largely on tangible objects used by a single person (the smart watch and reminder systems in the user study), however the principles of electronic storyboarding (active interpretation of annotated sketches) can be extended to other areas. In this section we outline three new applications of electronic storyboarding principles in fashion design, architecture, and human in the loop computing.

Fashion Design / Wearable Computing

As wearable computing becomes more prevalent and accessible, designers will seek to incorporate these technologies into their work. A fashion exhibition has been a main feature of the International Symposium on Wearable Computing for several years. Wearable computing presents unique wearability challenges that must balance user comfort [131] but effective sensor placement for accurate activity recognition. Figure 6.2 show the sketches of a designer working

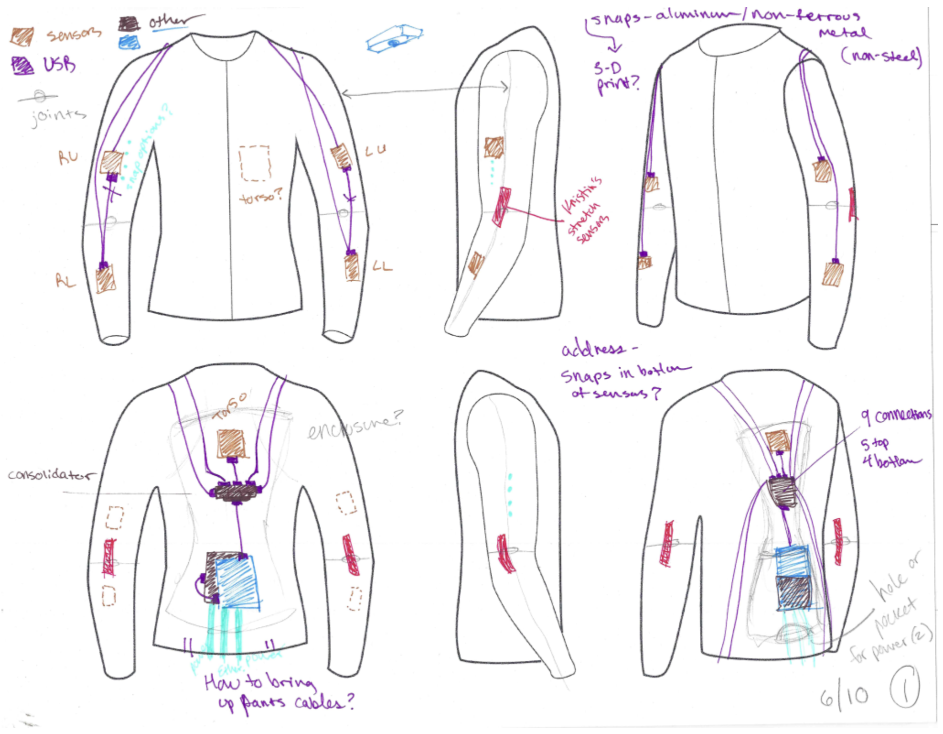

sensors
other
USB
joints
RU
RL
LU
LL
torso?
snaps - aluminum / non-ferrous metal (non-steel)
3-D print?
Kristin's stretch sensors
address - Snaps in bottom of sensors?
torso
enclosure?
consolidator
9 connections
5 top
4 bottom
hole or pocket for power (2)
How to bring up pants cables?
6/10
1

Figure 6.2: Designer sketches of a wearable computing system (courtesy of Mary Lee Carter).

through various placement locations for sensors and USB cables for connectivity. For a given application, the best placement of sensors can be known [149]. This application-specific information, and general placement guidelines, could be actively fed back to the designer to help them iterate more quickly through the potential design space. In a scenario such as this tagging may not be appropriate. Given a standard mannequin outline, the color coding of strokes (as the designer in Figure 6.2 has already done with purple for USB and grey for sensors) and the weight of those strokes could be used to infer computational and wearability information much in the same way tags inferred the underlying device state. The output of this process would not necessarily be code, but could be active suggestions or links to similar wearable systems to provide inspiration and new ideas.

Interactive Architecture

Prior to conducting the research for this dissertation the author was the an instructor for an interactive architecture studio for architecture and fashion design students. One of the challenges faced by students in the course was prototyping their projects in "real-life" scenarios. Many of the student projects were large installations that interacted with the environment and many users. While these applications could be prototyped on the table top, they were never able to be fully tested until installed in the final venue, which in this case was the lobby of the university architecture building. Figure 6.3 shows an example project that reacted to the ambient noise in the building. For many of the projects, the students could not replicate the final interactions until the project was "completed" and installed.

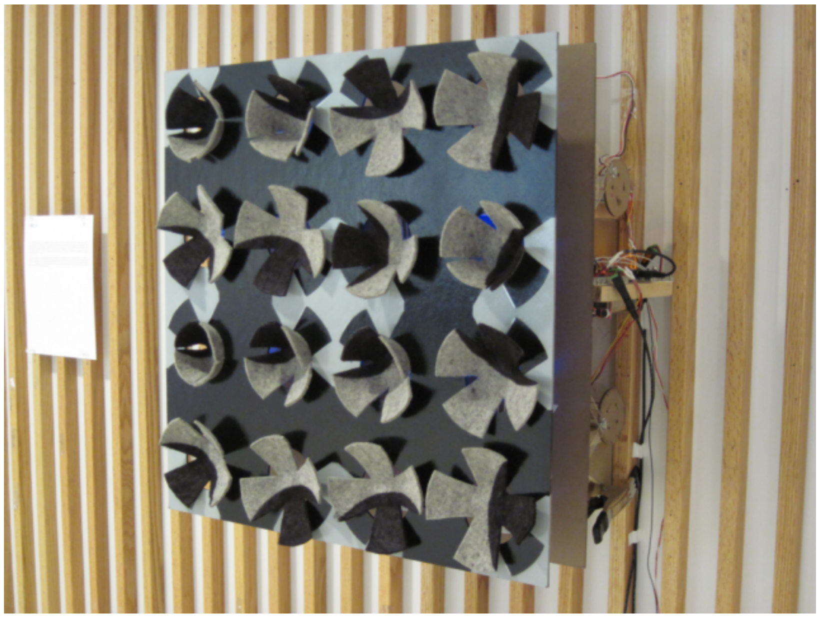


Figure 6.3: Student project from interactive architecture studio.

One possible solution for this problem could be to interpret architecture sketches to explore or simulate how interactive systems would respond in a certain space. Existing approaches have created critics that analyze sketches for architectural floorplans [150], furniture designs [151], and automatic retrieval of design plans. New approaches could use annotations to combine architecture sketches to automatically creative a simulation environment in which the interactive system can be tested. Architecture students were proficient at programming their systems using Firefly [152], which could then be coupled to the architecture simulation to test out their ideas before installation.

Resolving Specification Ambiguity using Humans in the Loop

One of our major contributions of this work is moving from informal design artifacts (storyboards) to more formal artifacts (source code, state charts, and textual descriptions). While this research was targeted as resolving pervasive computing systems, our work can generalize to more arbitrary scenarios where information must be resolved between two distinct sets of knowledge. In the remainder of this section we briefly describe a new application for distributed product design teams.

From our recent work in the National Science Foundation I-Corps program, we have discovered that large product design teams have difficulty communicating design changes between user experience, software engineering, and hardware development teams. In large companies these teams are often distributed and frequent communication is difficult or not well executed. For wearable, tangible, or experiential systems, changes in one aspect of the product impacts all parties. The hardware team may not understand how removing a sensor can impact the user experience, whereas the user experience team may not understand how demanding certain interactions are upon the hardware. While we have shown on a small scale how electronic storyboards coordinate between two participants, it could be extended to enable better coordination amongst these teams. An initial electronic storyboard could be created for the intended scenarios and discipline-specific changes by the parties could be queried against that scenario. By building meta-models of hardware, software, and user-experience concerns, these constraints could be centralized at the storyboard level to help teams interactively and iteratively understand the impact of their design changes.

Overall, our approaches can be applied where users must iteratively interrogate knowledge models. The mechanisms of user-applied annotations and natural language processing can be extended into new domains. The key insight is to keep the human in the loop to leverage their knowledge and iteratively remove ambiguity. It is not possible to create a model with all relevant domain knowledge - the user must come in at some point. The challenge is to determine what approaches create the most “crisp” specification that does not overly encumber the user.

6.3 Summary

Overall, the advancement of any creative discipline is related to the quality of its tools [153]. This dissertation has sought to advance the quality of and benefits provided by pervasive computing design tools. While optimized tools can always be created for specific domains or narrow applications, we have endeavored to create a general tool that can be used to address applications in a variety of domains. Electronic storyboards have been shown usable by design teams and helpful in expression design ideas and reflecting on storyboarded systems. The design space and future potential of this work is much larger than the work conducted and leaves areas for expansive future work.

Appendix A

Storyboard Compilation

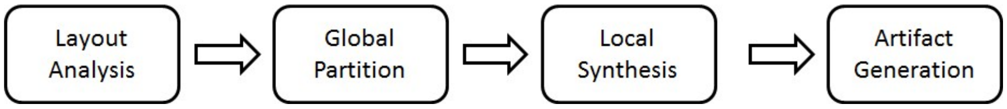
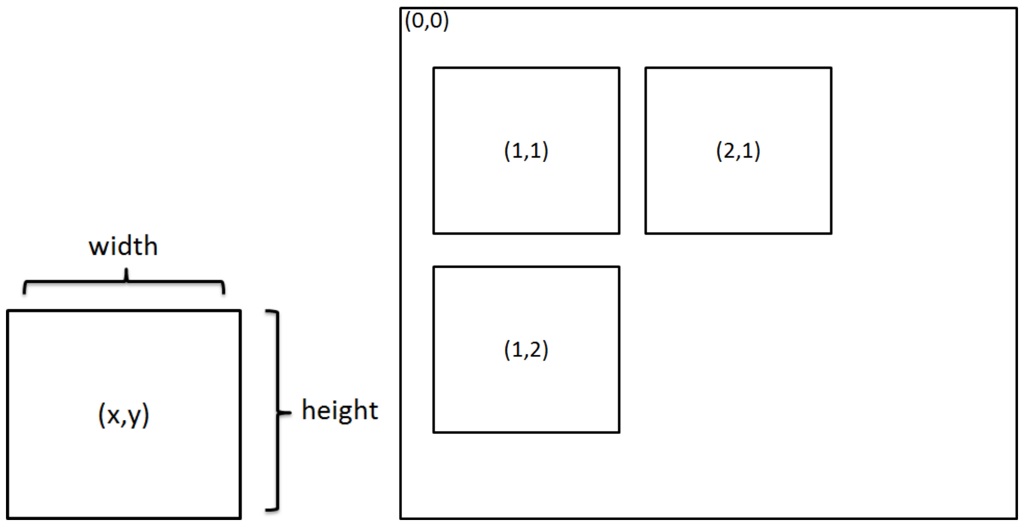


Figure A.1: Electronic Storyboard compilation process

This section provides detailed instructions on implementing the transformations within the electronic storyboarding tool. These descriptions were omitted from the main chapter for brevity and are produced here for completeness. Figure A.1 shows the four main stages of compilation. Section A.1 describes Layout Analysis which produces a graph that contains the structure of the storyboard. The next stage, Global Partitioning, divides the graph into regions of similar time and context. This process is described in Section A.2. Section A.3 describes Local Synthesis which searches each graph from the partition and transforms it into a timed automata. Finally, each automata can be transformed into several design artifacts, such as source code, state charts, and textual descriptions. This process was covered in Section 3.3.4.

In this section, the structure of the storyboard is referenced as graph. We defined a graph to be $G = (V, E)$ where V is a set of vertices $\{v_i, v_{i+1}...\}$ and where E is a set of directed edges that link vertices $\{(v_1, v_2), (v_1, v_3), ...\}$. For these graphs, the vertices are the individual frames in the storyboard and the edges are the implied, conditional, and message connectors of the storyboard. We will use the relationships between vertices and frames, and edges and connectors interchangeably. In this way vertices may contain annotations or other information that would be expected of frames.

A.1 Layout Analysis



(a) Frame geometry. (b) Storyboard layout.

Figure A.2: Model of storyboard layout

A few preliminaries before beginning this section, we assume the layout of a storyboard is a collection of frames. Each frame is rectangular with a width and height, and has a center point in 2-D space. The collection of frames forms the storyboard layout. We assume the upper, left-most corner of the storyboard is the coordinate $(0, 0)$, extending out positively in the x and y-axis. A *frame block* is a sequence of frames that are geometrically proximate and would be "read" as a single group in a storyboard. The blocks would be "read" from left to right, and top to bottom as described in Section 3.3.1. In addition to the frames, there are *conditional* connectors that connect frames within the same frame block where all State annotations in the block reference the same device. *Message* connectors connect frames blocks where the device referenced in the block is different.

The primary task of Layout Analysis is to transform the storyboard in a graph structure that can be more easily interpreted by following parts of compilation. Algorithm 1 outlines how Layout Analysis is performed. Algorithms 2 and 3 provide helper functions.

Algorithm 1 Layout Analysis: Overview

```
Input: frames - set of frames $\{f_i, f_{i+1}, ...f_n\}$
Input: conditionals - set of conditional connectors between frames $\{(f_i, f_j), (f_j, f_k)...\}$
Input: messages - set of message connectors between frames $\{(f_i, f_j), (f_j, f_k)...\}$
Output: graphs - a set of graphs derived from the storyboard $\{(V_a, E_a), (V_b, E_b)....\}$
  function LAYOUTANALYSIS(frames, conditionals, messages)
      marked ← ∅                                              ▷ set to collect frames assigned a block
      unmarked ← frames                                       ▷ set for unassigned frames
      blocks ← ∅                                              ▷ set of collected blocks
      while |unmarked| < |frames| do                          ▷ Create blocks from unmarked frames
          prime ← find frame closest to the upper-left corner
          block ← BUILDBLOCK(unmarked,prime)
          if block is not ∅ ∧ prime is not ∅ then
              blocks ← collected ∪ block
              marked ← marked ∪ {∀ frames in block}
              unmarked ← unmarked − {∀ frames in block}
          end if
      end while
      if ∃ a conditional connection between blocks of different devices then   ▷ Check State annotations in block
          return ∅
      end if
      if ∃ a message connection between blocks of same devices then           ▷ Check State annotations in block
          return ∅
      end if
      graphs ← ∅
      for all block in blocks do                              ▷ Create a graph from each block
          vertices ← make each frame in block a vertex
          for all frame $(f_i)$ in the block do               ▷ Create edges between adjacent frames
              $edges \leftarrow edges \cup (f_i, f_{i+1})$    ▷ Frames in local block are implicitly connected
          end for
          graphs ← graphs ∪ {vertices,edges}
      end for
      for all $graph_i, graph_j$ in graphs do                 ▷ Conditional connectors were ignored in BuildBlock
          if ∃ connection in the original storyboard between $graph_i, graph_j$ then   ▷ Repair connected blocks
              $graph_i \leftarrow (V_i \cup V_j, E_i \cup E_j \cup$ {connection} )     ▷ Merge vertices and edges of graphs
              $graphs \leftarrow graphs - graph_j$
          end if
      end for
  end function
```

Algorithm 2 Layout Analysis: Building blocks from unsorted frames

```
Input: frames - set of frames
Input: prime - a frame
Output: block - set of geometrically connected frames
  function BUILDBLOCK($frames$,$prime$)
      current ← prime                                         ▷ current frame of interest
      starter ← prime                                         ▷ frame at the start of my row
      collected ← $\emptyset$                                 ▷ ordered set to keep collected frames
      while current is not $\emptyset$ do
          collected ← collected ∪ current
          east ← FINDNEIGHBORATDIRECTION(current, frames, EAST)    ▷ Algorithm 3
          if east is $\emptyset$ then
              newStarter ← FINDNEIGHBORATDIRECTION(starter, frames, SOUTH)
              if newStarter is $\emptyset$ then                ▷ No south frame was found, return
                  return collected;
              else
                  starter ← newStarter;
                  current ← newStarter;
              end if
          else
              current ← east
          end if
      end while
  end function
```

Algorithm 3 Layout Analysis: Find closest neighbor at a direction

```
Input: target - frame to begin finding neighbors
Input: candidates - set of frames that are candidate neighbors
Input: direction - cardinal direction in which to search
Output: neighbor - neighbor at specified direction.
  function FINDNEIGHBORATDIRECTION(target,candidates,direction)
      t ← target
      C ← candidates
      neighbor ← $\emptyset$
      maxDist ← $3 * max(2 * \sum_{\forall c \in C} \frac{c.width}{||C||}, 2 * \sum_{\forall c \in C} \frac{c.height}{||C||})$    ▷ Maximum distance between frames
      ewLimit ← $0.1$                                  ▷ Empirically derived parameter for Eclipse
      nsLimit ← $20$                                   ▷ Empirically derived parameter for Eclipse
      if direction is EAST then
          select $c | \forall c \in C, min(dist(c,t) \wedge (c.x - target.x) > 0) \wedge |\frac{c.x-t.x}{c.y-t.y}| < ewLimit \wedge dist(c,t) < maxDist$
      else if direction is WEST then
          select $c | \forall c \in C, min(dist(c,t) \wedge (c.x - target.x) < 0) \wedge |\frac{c.x-t.x}{c.y-t.y}| < ewLimit \wedge dist(c,t) < maxDist$
      else if direction is NORTH then
          select $c | \forall c \in C, min(dist(c,t) \wedge (c.y - target.y) < 0) \wedge |\frac{c.x-t.x}{c.y-t.y}| > nsLimit \wedge dist(c,t) < maxDist$
      else if direction is SOUTH then
          select $c | \forall c \in C, min(dist(c,t) \wedge (c.y - target.y) > 0) \wedge |\frac{c.x-t.x}{c.y-t.y}| > nsLimit \wedge dist(c,t) < maxDist$
      else
          $c \leftarrow \emptyset$
      end if
      neighbor ← $c$
      return neighbor
  end function
```

A.2 Global Partition

In Section A.1 we outlined how the storyboard is transformed into a series of graphs based upon frame layout and connector information. In this section we describe how those graphs are partitioned into different regions of time and context. Global Partitioning is initiated in Algorithm 4. The function PartitionGraph (Algorithm 5) is repeatedly called to partition the graph into smaller graphs. Before returning, Algorithm 6 is called to heal global regions that might have been separated.

Algorithm 4 Global Partition: Overview

```
Input: graphs - set of graphs generated from Layout Analysis
Output: regions - set of regions where each region contains a storyboard graph, context information, and temporal
  information. (Graph, Context, Time)
  function GLOBALPARTITION(graphs)
      regions ← ∅
      for all graph in graphs do
          collected ← ∅
          marked ← ∅
          newRegions ← PARTITIONGRAPH(graph,null,null,collected,marked)          ▷ Algorithm 5
          regions ← regions ∪ newRegions
      end for
      healed ← COMBINEEQUIVALENTREGIONS(regions)          ▷ Repair regions that might have been separated
      return healed
  end function
```

Algorithm 5 Global Partition: Recursively partition a graph into subgraphs of different time and context

```
Input: graph - storyboard graph
Input: context - current context
Input: time - current time interval
Input: collected - collected regions
Input: marked - vertices that have been visited
  function PartitionGraph($graph, context, time, collected, markedNodes$)
      current ← select vertex in graph that has no incoming edges
      regionStart ← current
      if current ∈ markedNode then                        ▷ Previously visited this node, return
          return
      end if
      end ← false                                          ▷ If graph ends in this recursion level
      while end is false do
          if current ∈ marked then                         ▷ Previously visited this node, return
              end ← true
          else
              marked ← marked ∪ current
          end if
          justAdded ← false                                ▷ Have we just partitioned
          for all data elements in current do              ▷ Extract all the information from the frame
              contextChange ← true if data causes a context change, otherwise false
              temporalChange ← true if data cases a temporal change, otherwise false
              if contextChange is true or temporalChange is true then
                  subgraph ← subgraph of graph between regionStarter and current
                  collected ← collected ∪ (subgraph, context, time)
                  justAdded ← $true$
                  regionStart ← current
                  newContext ← is new context if contextChange is true, context otherwise
                  newTimeInterval ← is new temporal interval if temporalChange is true, time otherwise
              end if
          end for
          $edges$ ← select outbound edges from current     ▷ Determine next frame/vertex
          next ← ∅
          if $|edges|$ is 0 then                           ▷ End of graph
              collected ← collected ∪ (current,context,time)
              return
          else if $|edges|$ is 1 then
              $next$ ← select the destination of the edge
          else                                             ▷ There are multiple edges, recurse
              if $justAdded$ is $true$ then
                  collected ← (current,context,time)
              end if
              for all e ∈ edges do
                  subgraph ← create subgraph from all descendants starting at target
                  PartitionGraph(subgraph, context, time, collected, marked)     ▷ Recursion
              end for
              end ← true;
          end if
          current ← next
      end while
  end function
```

Algorithm 6 Global Partition: Merge graphs that may have been partitioned but describe similar time and context

```
Input: regions - set of context time regions where each region is (Graph, Context, Time)
Output: healedRegions - set of merged regions
  function CombineEquivalentRegions(regions)
      if |regions| is 1 then
          return regions
      end if
      merged ← ∅
      newRegions ← ∅
      for all $(r_a, r_b) \in$ regions do
          $r_a = (G_a, Context_a, Time_a)$
          $r_b = (G_b, Context_b, Time_b)$
          merged ← merged $\cup r_a \cup r_b$                                    ▷ mark as merged
          if $(Context_a, Time_a)$ is equivalent to $(Context_b, Time_b)$ and $G_b$ is reachable from $G_a$ then
              if $\exists r_c \in$ newRegions with same context and time then
                  $r_c = (G_c, Context_c, Time_c)$
                  $G_c = (G_a \cup G_b \cup G_c)$                ▷ merge these two regions into the existing one
              else
                  newRegions ← newRegions $\cup \{G_a \cup G_b, Context_a, Time_a\}$      ▷ add to collected regions
              end if
          end if
      end for
      for all $\forall r \in$ regions $|r \notin$ merged do                ▷ Put back everything that was not merged
          newRegions ← r $\cup$ merged
      end for
      return newRegions
  end function
```

A.3 Local Synthesis

Local Synthesis takes each region (Graph, Context, Time) generated by Global Partition and converts the graph in that region into a timed automaton. If there are multiple regions, and therefore multiple automata, each automata is examined for temporal relationships. The only relationships current supported are *before*, *after*, and *independent*. Automata with a *before* or *after* are scheduled such that one must complete before the other can begin. Automata with *independent* relationships are individual threads. Algorithm 7 presents the overall Local Synthesis process. Algorithm 8 builds an automaton from a graph, and Algorithm 9 attempts to build state transitions through interaction with the user.

Algorithm 7 Local Synthesis: Overview

```
Input: regions - device regions from Global Partition
  function LOCAL SYNTHESIS($regions$)
      synthesized ← ∅
      for all $r \in regions$ do
          graph ← select the graph from r
          starter ← select vertex from graph with no inbound edges.
          builder ← ∅
          edges ← ∅
          automaton ← ∅
          BUILDFROMGRAPH(region.Graph, starter, starter, builder, edges, automaton)
          automaton ← assign automaton Context and Time
          synthesized ← synthesized ∪ automaton
      end for
      for all $(automaton_i, automaton_j) \in synthesized$ do
          if $automaton_i$ is before $automaton_j$ then
              notify compiler to set flag for $automaton_i$ completion
          else if $automaton_i$ is after $automaton_j$ then
              notify compiler to set flag for $automaton_i$ completion
          else
              notify compiler to create independent threads for $automaton_i$ and $automaton_j$
          end if
      end for
      Generate code and state charts for each $automaton \in synthesized$        ▷ Section 3.3.4
  end function
```

Algorithm 8 Local Synthesis: transform a graph into a timed automaton

```
Input: graph - overall storyboard graph that is being parsed
Input: current - current vertex/frame that is being examined
Input: starter - vertex/frame that is the beginning of the current interval
Input: builder - state builder to hold annotation information
Input: marked - edges within the graph that have been followed.
Input: automaton - a timed automata of states and transitions
  function BUILDFROMGRAPH($graph, current, starter, builder, edges, automaton$)
      while current is not $\emptyset$ do
          data $\leftarrow$ extract State, Event, and Action information from current    ▷ See Section 3.3.3
          builder $\leftarrow$ add extracted data to builder to see if a transition is available
          newStatus $\leftarrow$ get the current status of builder
          if newStatus is not ERROR and newStatus is not INCOMPLETE then    ▷ If in a buildable state
              success $\leftarrow$ ATTEMPTBUILD(starter, current, builder, newStatus,automaton)    ▷ Attempt state transition
              if success is true then
                  starter $\leftarrow$ current
                  newBuilder $\leftarrow$ create new State Builder
                  lastState $\leftarrow$ get the destination state from current
                  newBuilder $\leftarrow$ add lastState as the starting state
                  newBuilder $\leftarrow$ add data in current that was not used in the build    ▷ Add leftover data
              end if
          end if
          edges $\leftarrow$ select outbound edges from current
          if |edges| is $\emptyset$ then
              postBuild $\leftarrow$ get the current status of builder
              if postBuild is not ERROR and postBuild is not INCOMPLETE then
                  ATTEMPTBUILD(starter,current,builder,postBuild,automaton)    ▷ Clean up before exiting
              end if
              current $\leftarrow \emptyset$
          end if
          if |edges| $> 0$ then    ▷ Multiple edges, recurse
              for all e in edges where $e \notin marked$ do
                  marked $\leftarrow$ marked $\cup$ e
                  newBuilder $\leftarrow$ clone existing builder    ▷ Save builder state
                  if e is a conditional connection and is labeled then
                      newEvent $\leftarrow$ create new event from labeled connection
                      newBuilder $\leftarrow$ add newEvent to newBuilder
                  end if
                  destination $\leftarrow$ get destination vertex/frame from e
                  BUILDFROMGRAPH(graph, destination, starter, newBuilder, marked)    ▷ Recurse
              end for
          end if
          if |edges| is 1 then
              marked $\leftarrow$ marked $\cup$ edges
              current $\leftarrow$ get destination vertex/frame from edges
          end if
      end while
  end function
```

Algorithm 9 Local Synthesis: attempt to build a state transition

```
Input: begin - first vertex/frame in the graph to examine
Input: end - last vertex/frame in the graph to examine
Input: builder - state builder to hold data from storyboard
Input: status - current builder status
Input: automaton - timed automaton to add transition
Output: success - true or false based upon transition building
  function AttemptBuild($begin, end, builder, status, automaton$)
    newTransition ← ∅
    switch status do
      case Complete                          ▷ All information is available for a transition
        newTransition ← (builder.initialState, builder.destinationState, builder.Event, builder.Action)
      case Missing Action       ▷ Action is missing, may cascade to Multiple Events if successful
        action ← query user to select NLP for Action or manually specify.
        newTransition ← (builder.initialState, builder.destinationState, builder.Event, action)
      case Missing Event        ▷ Event is missing, may cascade to Multiple Actions if successful
        event ← query user to select NLP for Event or manually specify
        newTransition ← (builder.initialState, builder.destinationState, event, builder.Action)
      case Two States                          ▷ Event and Action information are missing
        event ← query user to select from NLP or manually specify an Event
        action ← query user to select from NLP or manually specify action based upon event
        newTransition ← (builder.initialState, builder.destinationState, event, action)
      case Multiple Events                     ▷ Multiple events with single action available
        event ← query user to select an event that causes action or manually specify.
        newTransition ← (builder.initialState, builder.destinationState, event, builder.Action)
      case Multiple Actions                    ▷ Multiple actions with a single event available
        action ← query user to select an action that is caused by event
        newTransition ← (builder.initialState, builder.destinationState, builder.Event, action)
      case Multiple Events and Actions   ▷ Multiple events and actions are available with two unique states
        event ← query user to select an event that causes the state transition
        action ← query user to select an action that causes event
        newTransition ← (builder.initialState, builder.destinationState, event, action)
      case Self-Transition Available           ▷ If single Event, Action, and State are available
        query ← ask user if this is event causes action
        if query is true then
          newTransition ← (builder.initialState, builder.initialState, event, action)
        end if
    if newTransition is not ∅ then
      automaton ← add newTransition to automaton
      builder ← remove data from builder that was used in newTransition
      return true
    else
      return false
    end if
  end function
```

Appendix B

Development History of Electronic Storyboards

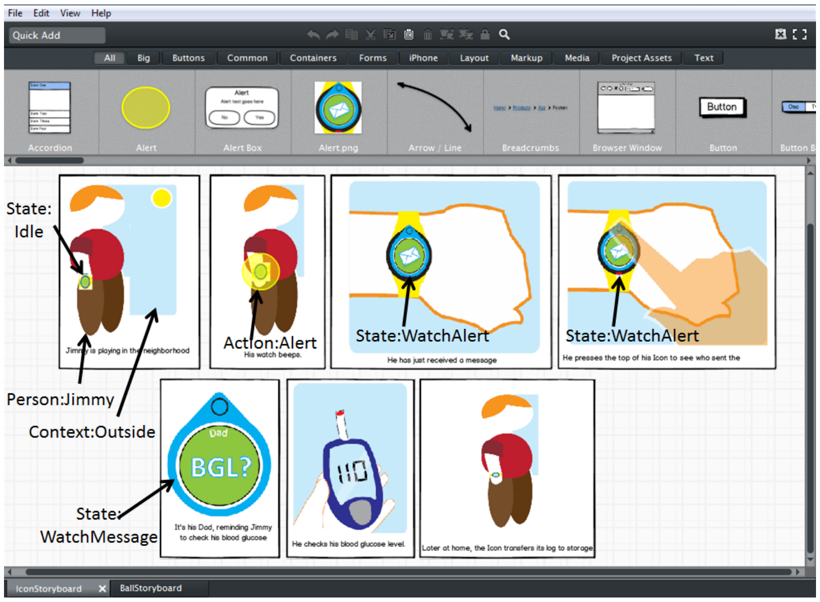
B.1 Overview

In Chapter 3 we described the high-level design challenges in developing Electronic Storyboards. In this Appendix we provide more detail about the evolution and design choices of the tool itself. Section B.2 describes the evolution of the tool environment for electronic storyboards and describes different approaches that were attempted. Section B.3 describes various warnings and errors messages from the storyboard compiler that can be presented to the user.

B.2 Tool Environment and User Input

B.2.1 Version A: Balsamiq Mockups

The first prototype of electronic storyboards was created in Balsamiq Mockups [154]. This tool enables designers to rapidly create mockups of user interfaces through a selection of existing media and icons. Balsamiq was used to replicate existing storyboards from a product design class and test the underlying feasibility of our approach. An example of one of these Balsamiq storyboards is shown in Figure B.1. Balsamiq was advantageous because each media element in the storyboard (such as the images shown in Figure B.1) could be tagged with meta information and the entire storyboard could be exported as an XML file. This XML file was parsed by an initial version of the storyboard complier written in Java. This process proved out the feasibility of our approach and was used to create the work in [89].


File Edit View Help
Quick Add
All Big Buttons Common Containers Forms iPhone Layout Markup Media Project Assets Text
Accordion Alert Alert Box Alert.png Arrow / Line Breadcrumbs Browser Window Button
State: Idle
Action:Alert
State:WatchAlert
State:WatchAlert
Person:Jimmy
Context:Outside
State: WatchMessage
BGL?
IconStoryboard
BallStoryboard

Jimmy is playing in the neighborhood

His watch beeps.

He has just received a message

He presses the top of his Icon to see who sent the

It's his Dad, reminding Jimmy to check his blood glucose

He checks his blood glucose level.

Later at home, the Icon transfers its log to storage

Figure B.1: Initial electronic storyboard in Balsamiq

B.2.2 Version B: Java + Swing GUI

The initial tool development in Balsamiq presented several difficulties. First, media to create the images for the storyboard had to be imported manually by the user. If users wanted to maintain the same level of fidelity as in Figure B.1 they were required to create the graphics beforehand, import them, and re-arrange to fit into place on the screen. This proved time consuming and difficult. Second, Balsamiq was not consistent in the structure of the generated XML. This made parsing the storyboard in the compiler difficult and information was frequently lost. To address these issues a customized GUI was developing within Java Swing that would be the intended tool environment. This approach offered freedom to create a tool that we desired but presented significant overhead. Simple actions such as Save, Load, and Undo had to be manually implemented. This version of electronic storyboards was short-lived and was quickly replaced by the Eclipse GEF version described in Chapter 3.

B.2.3 Version C: Eclipse Graphical Editor Framework

The current version of electronic storyboards is shown in Figure B.2 and described more fully in Chapter 3. Several design changes were intentional to this version. First, the use of clip art and custom media as in the previous versions was removed. While the storyboard in Figure B.1 look clean they are time consuming to create. For storyboards of that fidelity, graphics had to be created in Illustrator (or similar tools) and then be imported into the electronic storyboard. If graphics did not align on the screen or present well visually, they had to be edited. We attempted to enable direct sketching of graphics in Eclipse to remedy this problem. However this ability was difficult to implement

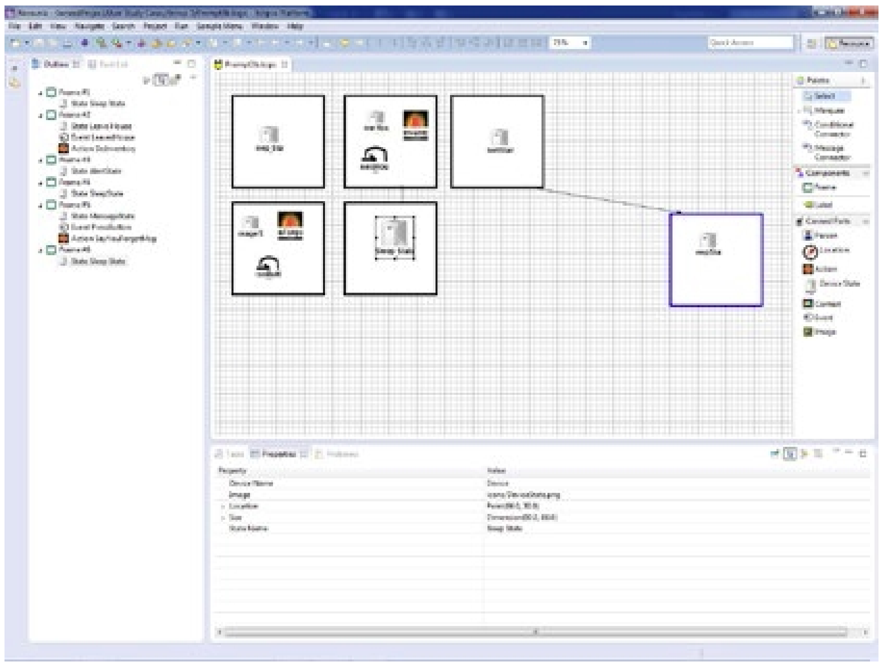


Figure B.2: Current version of electronic storyboards.

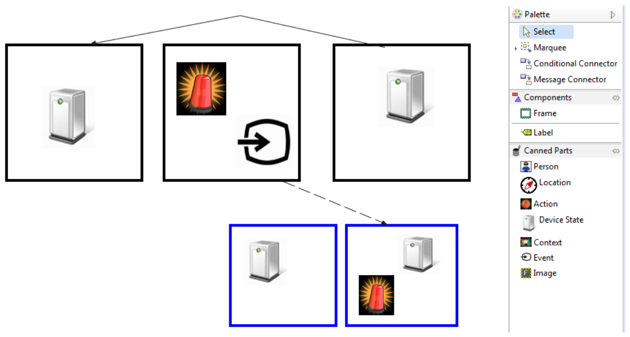
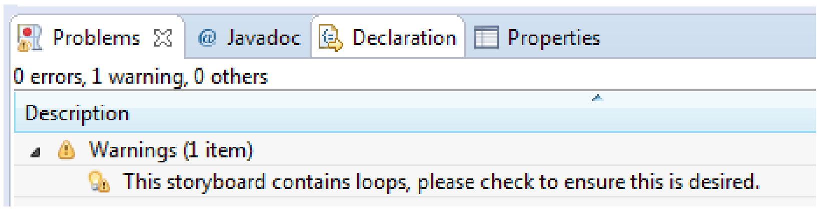


Figure B.3: Icon-based storyboard in Eclipse GEF.

and no existing libraries could be found. Thus the icon approach of Figure B.3 was adopted where each icon on the screen indicates specific tagged information relevant to the story.

This version retains the ability to import custom graphics like previous versions, however the underlying software is much more powerful. Basic GUI actions such as Save, Load, Undo, and movement of graphics are provided for "free". Furthermore, the interface is integrated into Eclipse and can be directly accessed by the compiler unlike Version A which required two separate programs. We will continue to use the icon-based approach until a suitable sketching tool can be found. One candidate is Cog Sketch [155]. However, continued work in Eclipse GEF provides additional abilities that are considered in the next section.



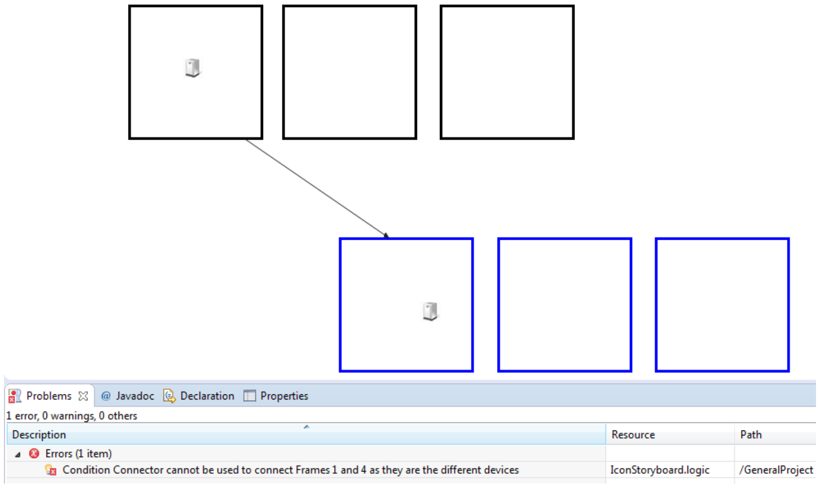
0 errors, 1 warning, 0 others

Description

Warnings (1 item)

This storyboard contains loops, please check to ensure this is desired.

Figure B.4: Eclipse warning about loops in the storyboard.



1 error, 0 warnings, 0 others

Description	Resource	Path
Errors (1 item)		
Condition Connector cannot be used to connect Frames 1 and 4 as they are the different devices	IconStoryboard.logic	/GeneralProject

Figure B.5: Message Connector Error

B.3 Errors and Warnings

One of the primary challenges in creating electronic storyboards was translating model-driven engineering concerns to the user level. Layout analysis is the most challenging part of the storyboard compiler because of the varieties of potential storyboard layouts. The use of connectors and different frame sizes makes deriving the proper storyboard graphs difficult. Several methods were attempted to make it easier for the user (and consequently the compiler) to process storyboards. First, we used the message feature of Eclipse to provide errors and warnings about critical elements. Storyboard loops are handled by the compiler but can be problematic at times. Figure B.4 shows a warning provided to the user to ensure that the loops detected within the storyboard are actually intended.

Another difficulty is communicating the difference between message connectors and conditional connectors in the storyboard. Where there are multiple devices in a storyboards only message connectors can link the frames of each device. In the example storyboard in Figure B.5, a message connector (solid arrow) is used to connect frames that describe interactions with two different devices. A similar error message would be present if a message connector was used to link frames that describe the same device.

Finally, electronic storyboards use a geometry-based approach to determine where sets of frames should be grouped together. An explanation of this approach can be found in Section 3.3.1 and the exact implementation in

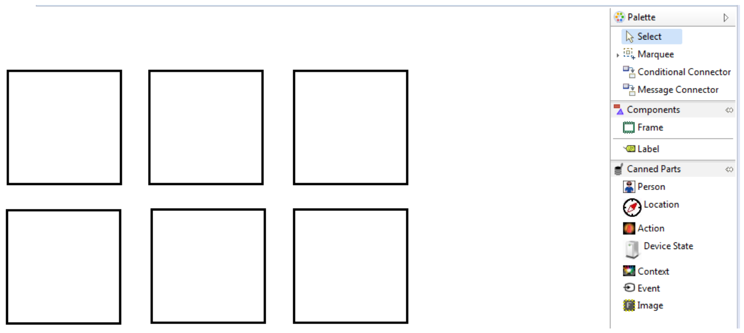


Figure B.6: Single frame block.

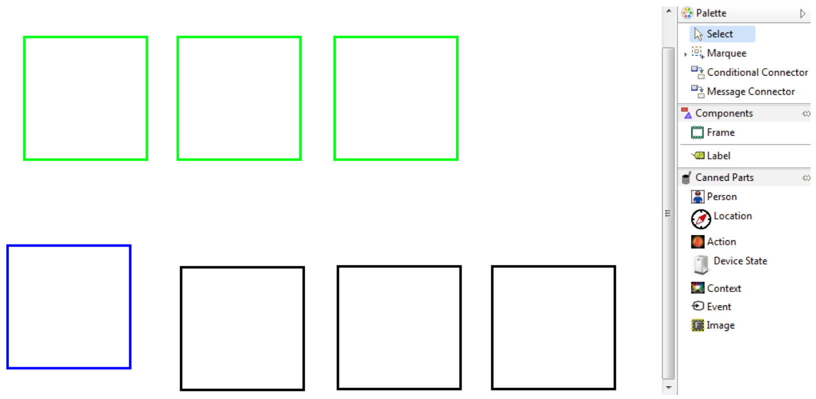


Figure B.7: Multiple frame blocks detected.

Section A.1. The challenge of layout analysis is to communicate to the user which frames the tool has grouped together. The algorithm may group or ungroup frames that the user did not intend. We address this approach by assigning a color to the frames of each group while the storyboard is being created.

Figure B.6 shows a group of frames that are all colored black. Here the tool has grouped those frames together which is indicated by their same color. Figure B.7 shows the changes when a new frame is added. Three frame blocks are now detected. This information is important to the user as independent frame blocks are parsed as unique behaviors within the storyboard (see Local Synthesis in Section 3.3.1). If the user intends the storyboard to be one experience these frames should be moved around or connected with conditional connectors to indicate that relationship.

Appendix C

Publications and External Funding

The following publications and external funding were derived from this work:

1. Tom Martin (PI), Jason Forsyth, Christian James, "Storycoding I-Corps Team", National Science Foundation. Award Date: 4/1/2015. Amount: $50k.

2. Jason Forsyth, Tom Martin, "Extracting Behavioral Information from Electronic Storyboards", Proceedings of the 6th ACM SIGCHI Symposium on Engineering Interactive Computer Systems, Rome, Italy, June 17-20, 2014

3. Jason Forsyth, "Using Electronic Storyboards to Support Interdisciplinary Design of Pervasive Computing Systems", Poster Presentation for the Sixth Annual Ph.D. Forum on Pervasive Computing and Communications (PerCom), San Diego, March 20th, 2013

4. Jason Forsyth, Tom Martin, "Tools for Interdisciplinary Design of Pervasive Computing", International Journal of Pervasive Computing and Communications, vol. 8, no. 2, pp. 112-132, June 2012

Appendix D

IRB Approval

This Appendix includes the IRB approval letter authorizing the user study.

VirginiaTech

Office of Research Compliance
Institutational Review Board
North End Center, Suite 4120, Virginia Tech
300 Turner Street NW
Blacksburg, Virginia 24061
540/231-4606 Fax 540/231-0959
email irb@vt.edu
website http://www.irb.vt.edu

MEMORANDUM

DATE:	January 29, 2014
TO:	Thomas L Martin, Jason Brinkley Forsyth
FROM:	Virginia Tech Institutional Review Board (FWA00000572, expires April 25, 2018)
PROTOCOL TITLE:	Studying the Effectiveness of Electronic Storyboards in Rapid Prototyping
IRB NUMBER:	**13-677**

Effective January 28, 2014, the Virginia Tech Institution Review Board (IRB) Chair, David M Moore, approved the Amendment request for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report within 5 business days to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

http://www.irb.vt.edu/pages/responsibilities.htm

(Please review responsibilities before the commencement of your research.)

PROTOCOL INFORMATION:

Approved As:	**Expedited, under 45 CFR 46.110 category(ies) 6,7**
Protocol Approval Date:	**August 14, 2013**
Protocol Expiration Date:	**August 13, 2014**
Continuing Review Due Date*:	**July 30, 2014**

*Date a Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

FEDERALLY FUNDED RESEARCH REQUIREMENTS:

Per federal regulations, 45 CFR 46.103(f), the IRB is required to compare all federally funded grant proposals/work statements to the IRB protocol(s) which cover the human research activities included in the proposal / work statement before funds are released. Note that this requirement does not apply to Exempt and Interim IRB protocols, or grants for which VT is not the primary awardee.

The table on the following page indicates whether grant proposals are related to this IRB protocol, and which of the listed proposals, if any, have been compared to this IRB protocol, if required.

Appendix E

Copyright Statements

This Appendix contains copyright permission for content used in this dissertation. Chapter 2 contains work published under Emerald Group Publishing. Permission is retained for use in this dissertation and future works. Chapter 3 contains work published under the Association of Computing Machinery. Permission is retained for use in the dissertation and future works. Figure 6.2 is the work of Mary Lee Carter. Permission has been obtained for use in this and future works. Figure 2.3 is adapted from [1]. Fair use analysis for that figure is included at the end of this Appendix.

Login

Username:
Password:
Login

- OpenAthens login
- Shibboleth login
- Forgot password?

Welcome:
Guest

Product Information:-

For Journals
For Books
For Case Studies
Regional information

Services

Publishing Services
Research and Publishing Pathway
Event Services

Resources:

Licensing Solutions
For Authors
Impact of Research
Writing for Emerald
Editing Service
Emerald Literati Network
Guide to Getting Published
How To Guides
Editor Interviews
Emerald and Open Access
For Librarians
For Engineers
Research Zone
Learning Zone
Teaching Zone
Multimedia Zone

Emerald | Insight
Visit Emerald's dedicated research platform
Search & browse

Home > For Authors > Writing for Emerald > Emerald Author rights

Emerald Author rights

Emerald understands that what you can and can't do with your work can be a maze for authors and different for each publishing house. The following page is designed to clarify Emerald's policies; however, if you still have any questions, please don't hesitate to contact permissions@emeraldinsight.com.

Definitions

Version of article or chapter	Definition
Pre-print (sometimes known as "Version 1")	The version of the work as submitted to the journal or book series (prior to the peer review process).
Post-print (sometimes known as "Version 2")	The version of the work as accepted for publication (this may include any amendments suggested as a result of the peer review process). This is the version that will be EarlyCited (journals only).
Version of Record (sometimes known as "Publisher PDF" or "Version 3")	The version of the work that appears in the official Emerald publication, which includes Emerald branding and formatting.

What are my rights as an Emerald author?

Where possible, Emerald seeks to obtain copyright for the material it publishes, without authors giving up their moral or scholarly rights to reuse their work. Read our FAQs.

Moral rights

You retain your statutory right to be recognized and fully attributed as the author of your work. Emerald asks all authors to assert this right as a standard clause on all our licence to publish forms. This right is protected by section(s) 77 and 78 of the UK Copyright, Designs and Patents Act (1988).

Retained rights

As an Emerald author, you retain the rights to reuse your work for a variety of scholarly purposes. Please note that uses permitted below are subject to full referencing and credit of original publication:

		Version	
	Pre-print	**Post-print**	**Version of Record**
Make photocopies for teaching and classroom purposes.	✔ Unlimited	✔ Unlimited	✔ Unlimited
Make photocopies for conferences or as handouts.	✔ Unlimited	✔ Unlimited	✔ Up to 25 copies. If you require more than 25 copies, please contact permissions@emeraldinsight.com
Inclusion in your dissertation or thesis	✔	✔	✔ This version may be included in the print version of your thesis/dissertation. If an electronic deposit is required, this must be the pre- or post-print version.
Reuse figures or extracts from your article/chapter in other works authored by you.	✔	✔	✔
Reuse or republish the entirety of your article/chapter in any future works.		✔ Providing that you are the named author/editor of the new work, i.e. your name will appear on the front cover of the new work.	
Deposit of your article in your institutional repository	✔ Immediately post-official publication. Please see below for 'Conditions of deposit.'	✔ Please see below for 'Conditions of deposit.'	x
Deposit of your article on your faculty, personal or corporate website	✔ Immediately post-official publication. Please see below for 'Conditions of deposit.'	✔ Please see below for 'Conditions of deposit.'	x
Deposit of your chapter in your institutional repository	✔ Immediately post-official publication.	✔ Immediately post-official publication.	x
Deposit of your chapter in your faculty, personal or corporate website	✔ Immediately post-official publication	✔ Immediately post-official publication	x
Sending your article/chapter via e-mail	x	x	x
Inclusion of your article/chapter on a VLE or Blackboard	✔ If your institution has a subscription to the content in question, you should include a URL back to the work on www.emeraldinsight.com. This will enable Emerald to remain COUNTER-compliant.		x

Table 1: Emerald Author reuse rights

The original Owner/Author permanently holds these rights:

- All other proprietary rights not granted to ACM, including patent or trademark rights.
- Reuse of any portion of the Work, without fee, in any future works written or edited *by the Author***, including books, lectures and presentations in any and all media.
- Create a "Major Revision" which is *not* subject to any rights in the original that have been granted to ACM
- Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, or (3) any repository legally mandated by an agency funding the research on which the Work is based.
- Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;
- Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version") to non-peer reviewed servers;
- Make distributions of the final published Version of Record internally to the Owner's employees, if applicable;
- Bundle the Work in any of Owner's software distributions; and
- Use any Auxiliary Material independent from the Work.

Additionally, authors who choose the ACM Publishing License Agreement, hold all other rights not granted to ACM in the License including the ownership of the copyright of the work.

Authors or their employers may retain copyright to embedded images (e.g., figures) with independent artistic value. Authors must grant permission for ACM to use the image in the context of the article in current and future formats. Such images must be declared at the time of article copyright transfer or grant of license, and declaration of copyright must be included within the image or the caption.

Re-use of third-party material contained in ACM published works always requires the consent of the copyright holder, as ACM's copyright or license does not cover third-party material.

In connection with any use by the Owner of the Definitive Version of Record, Owner should include the ACM citation and ACM Digital Object Identifier.

In connection with any use by the Owner of the Submitted Version (if accepted) or the Accepted Version or a Minor Revision, Owner shall use best efforts to display the ACM citation, along with a statement substantially similar to the following:

"© {Owner/Author | ACM} {Year}. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in {Source Publication}, http://dx.doi.org/10.1145/{number}."

**Requests made on behalf of others, i.e., for contributions to the work of other authors or other editors, may require payment of the fee.

2.6 Fixity of Works

ACM does not alter works once published. There are times, however, when it is appropriate to publish a revised or corrected version of a work; doing so requires the approval of the responsible editor.

Subject: Re: e-textiles sketches
From: Mary Lee Carter <marylee9@vt.edu>
Date: 03/27/2015 03:39 PM
To: Jason Forsyth <jforsyth@vt.edu>

I accept this agreement!

Mary Lee Carter
March 27 2015

On Mar 27, 2015, at 12:09 AM, Jason Forsyth wrote:

> Mary Lee,
>
> During the summer you created some sketches of wearable systems that I would like to use in future works. I have mentioned this to you previously but I would like to formalize this agreement so I don't have to ask you every time. I've attached an agreement that provides myself permission to use these sketches in future work. Most notably in my upcoming dissertation.
>
> Your agreement or disagreement on this matter will have no impact on any role, position, or anything otherwise. I appreciate your work and feel it could be very helpful in the future.
>
> If this is acceptable to you could you: (1) print out the first page, (2) scan it, and (3) email it back to me?
>
> Thanks,
>
> Jason
> <ML Scans Agreement.pdf>

Mary Lee Carter

Virginia Tech Industrial Design
Coroflot.com/MaryLeeC/portfolio

Alpha Rho Chi, Metagenes
Kappa Kappa Gamma, Zeta Mu
VT Sport Clubs, Supervisor
Hokie Ambassadors

Draft 09/01/2009

(Questions? Concerns? Contact Gail McMillan, Director of the Digital Library and Archives at Virginia Tech's University Libraries: gailmac@vt.edu)

(Please ensure that Javascript is enabled on your browser before using this tool.)

Virginia Tech ETD Fair Use Analysis Results

This is not a replacement for professional legal advice but an effort to assist you in making a sound decision.

Name: Jason Forsyth

Description of item under review for fair use: Figure 24 in Sketching the User Experience: Getting the Design Right and the Right Design, Bill Buxton, Morgan Kaufmann, cited as Figure 2.3 in my work.

Report generated on: 05-20-2015 at : 10:55:41

Based on the information you provided:

Factor 1

Your consideration of the purpose and character of your use of the copyright work weighs: ***in favor of fair use***

Factor 2

Your consideration of the nature of the copyrighted work you used weighs: ***in favor of fair use***

Factor 3

Your consideration of the amount and substantiality of your use of the copyrighted work weighs: ***in favor of fair use***

Factor 4

Your consideration of the effect or potential effect on the market after your use of the copyrighted work weighs: ***in favor of fair use***

Based on the information you provided, your use of the copyrighted work weighs: *in favor of fair use*

Appendix F

Bibliography

[1] B. Buxton, *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, 2007.

[2] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, vol. 36, pp. 75–84, July 1993. [Online]. Available: http://doi.acm.org/10.1145/159544.159617

[3] T. Martin, K. Kim, J. Forsyth, L. McNair, E. Coupey, and E. Dorsa, "Discipline-based instruction to promote interdisciplinary design of wearable and pervasive computing products," *Personal and Ubiquitous Computing*, pp. 1–14, December 2011.

[4] E. Y.-L. Do and M. D. Gross, "Environments for creativity: a lab for making things," in *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*. New York, NY, USA: ACM, 2007, pp. 27–36.

[5] B. Moggridge, *Designing Interactions*. MIT Press, 2007.

[6] S. Dow, T. S. Saponas, Y. Li, and J. A. Landay, "External representations in ubiquitous computing design and the implications for design tools," in *Proceedings of the 6th conference on Designing Interactive Systems*, June 2006, pp. 241–250.

[7] M. Haensen, "User-centered process framework and techniques to support the realization of interactive systems by multi-disciplinary teams," Ph.D. dissertation, Universiteit Hasselt, 2011.

[8] C. Van der Lelie, "The value of storyboards in the product design process," *Personal and Ubiquitous Computing*, vol. 10, no. 2-3, pp. 159–162, 2006.

[9] E. Welbourne, M. Balazinska, G. Borriello, and J. Fogarty, “Specification and verification of complex location events with panoramic,” in *Pervasive Computing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6030, pp. 57–75.

[10] M. Shin, B. soo Kim, and J. Park, “Ar storyboard: An augmented reality based interactive storyboard authoring tool,” *IEEE / ACM International Symposium on Mixed and Augmented Reality*, vol. 0, pp. 198–199, 2005.

[11] B. P. Bailey, J. A. Konstan, and J. V. Carlis, “DEMAIS: designing multimedia applications with interactive storyboards,” in *Proceedings of the ninth ACM international conference on Multimedia*. New York, NY, USA: ACM, 2001, pp. 241–250.

[12] D. Pizzi, J. Lugrin, A. Whittaker, and M. Cavazza, “Automatic generation of game level solutions as storyboards,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 3, pp. 149 –161, sept. 2010.

[13] D. A. Schon and G. Wiggins, “Kinds of seeing and their functions in designing,” *Design Studies*, vol. 13, no. 2, pp. 135 – 156, 1992.

[14] Eclipse Graphical Editor Framework, http://www.eclipse.org/gef/.

[15] G. D. Abowd, “What next, ubicomp?: celebrating an intellectual disappearing act,” in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM, 2012, pp. 31–40.

[16] M. Weiser, “Designing calm technology,” http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm, 1995.

[17] Y. Liu, J. Goncalves, D. Ferreira, S. Hosio, and V. Kostakos, “Identity crisis of ubicomp?: mapping 15 years of the field’s development and paradigm change,” in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2014, pp. 75–86.

[18] M. Famelis, R. Salay, A. Di Sandro, and M. Chechik, “Transformation of models containing uncertainty,” in *Model-Driven Engineering Languages and Systems*. Springer, 2013, pp. 673–689.

[19] J. Forsyth and T. Martin, “Tools for interdisciplinary design of pervasive computing,” *International Journal of Pervasive Computing and Communications*, vol. 8, pp. 112 – 132, June 2012.

[20] M. Weiser, “The computer for the 21st century,” *Scientific American*, vol. 265, pp. 19–26, September 1991.

[21] M. Satyanarayanan, “Pervasive computing: vision and challenges,” *IEEE Personal Communications*, vol. 8, no. 4, pp. 10 –17, aug. 2001.

[22] M. Weiser, “The world is not a desktop,” *Interactions*, vol. 1, no. 1, pp. 7–8, Jan. 1994.

[23] N. Davies and H.-W. Gellersen, “Beyond prototypes: challenges in deploying ubiquitous systems,” *IEEE Pervasive Computing*, vol. 1, no. 1, pp. 26 – 35, Jan-Mar 2002.

[24] M. Ashby and K. Johnson, *Materials and Design - The Art and Science of Material Selection in Product Design*, 2nd ed. Butterworth-Heinemann, 2010.

[25] D. A. Schon, “Designing as reflective conversation with the materials of a design situation,” *Research in Engineering Design*, vol. 3, no. 3, pp. 131–147, 1992.

[26] D. Schon, *The Reflective Practitioner: How Professionals Think In Action*. Basic Books, 1984.

[27] G. Fischer, K. Nakakoji, J. Ostwald, G. Stahl, and T. Sumner, “Embedding critics in design environments,” *The knowledge engineering review*, vol. 8, no. 04, pp. 285–307, 1993.

[28] E. S. Poole, C. A. Le Dantec, J. R. Eagan, and W. K. Edwards, “Reflecting on the invisible: understanding end-user perceptions of ubiquitous computing,” in *Proceedings of the 10th international conference on Ubiquitous computing*. New York, NY, USA: ACM, 2008, pp. 192–201. [Online]. Available: http://doi.acm.org/10.1145/1409635.1409662

[29] T. Martin, K. Kim, J. Forsyth, L. McNair, E. Coupey, and E. Dorsa, “An interdisciplinary undergraduate design course for wearable and pervasive computing products,” in *15th Annual IEEE International Symposium on Wearable Computers (ISWC)*, June 2011, pp. 61 – 68.

[30] N. Cross, *Designerly ways of knowing*. Springer, 2006.

[31] B. R. Lawson, “Cognitive strategies in architectural design,” *Ergonomics*, vol. 22, no. 1, pp. 59–68, 1979.

[32] S. P. Dow, K. Heddleston, and S. R. Klemmer, “The efficacy of prototyping under time constraints,” in *Proceedings of the Seventh ACM Conference on Creativity and Cognition*, 2009, pp. 165–174.

[33] M. Buchenau and J. F. Suri, “Experience prototyping,” in *Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques*. New York, NY, USA: ACM, 2000, pp. 424–433.

[34] “Processing.” [Online]. Available: http://www.processing.org/

[35] Arduino, www.arduino.cc.

[36] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee, “Reflective physical prototyping through integrated design, test, and analysis,” in *Proceedings of the 19th annual ACM symposium on User interface software and technology*, 2006, pp. 299–308.

[37] Obrenovic, Željko and Martens, Jean-Bernard, "Sketching interactive systems with sketchify," *ACM Trans. Comput.-Hum. Interact.*, vol. 18, pp. 4:1–4:38, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1959022.1959026

[38] Y. Li and J. A. Landay, "Activity-based prototyping of ubicomp applications for long-lived, everyday human activities," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, April 2008, pp. 1303–1312.

[39] Y. Li, J. I. Hong, and J. A. Landay, "Topiary: a tool for prototyping location-enhanced applications," in *Proceedings of the 17th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2004, pp. 217–226. [Online]. Available: http://doi.acm.org/10.1145/1029632.1029671

[40] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay, "Papier-mache: toolkit support for tangible input," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2004, pp. 399–406.

[41] R. Hague, P. Robinson, and A. Blackwell, "Towards ubiquitous end-user programming," in *Proceedings of Ubicomp 2003*, 2003.

[42] R. van Herk, J. Verhaegh, and W. F. Fontijn, "Espranto sdk: an adaptive programming environment for tangible applications," in *Proceedings of the 27th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2009, pp. 849–858.

[43] G. Ngai, S. C. Chan, V. T. Ng, J. C. Cheung, S. S. Choy, W. W. Lau, and J. T. Tse, "i*catch: a scalable plug-n-play wearable computing framework for novices and children," in *Proceedings of the 28th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2010, pp. 443–452. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753393

[44] A. Warth, T. Yamamiya, Y. Ohshima, and S. Wallace, "Toward a more scalable end-user scripting language," in *Creating, Connecting and Collaborating through Computing, 2008. C5 2008. Sixth International Conference on*, jan. 2008, pp. 172 –178.

[45] E. Baafi and A. Millner, "A toolkit for tinkering with tangibles and connecting communities," in *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*. New York, NY, USA: ACM, 2011, pp. 349–352.

[46] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu, "a cappella: programming by demonstration of context-aware applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2004, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/985692.985697

[47] A. Parkes and H. Ishii, "Bosu: a physical programmable design tool for transformability with soft mechanics," in *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. New York, NY, USA: ACM, 2010, pp. 189–198. [Online]. Available: http://doi.acm.org/10.1145/1858171.1858205

[48] K. Truong, E. Huang, and G. Abowd, "Camp: A magnetic poetry interface for end-user programming of capture applications for the home," in *Proceedings of Ubicomp 2004*, 2004, pp. 143–160.

[49] M. H. Vastenburg, H. Fjalldal, and C. van der Mast, "Ubi-designer: a web-based toolkit for configuring and field-testing ubicomp prototypes," in *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*. New York, NY, USA: ACM, 2009, pp. 32:1–32:6. [Online]. Available: http://doi.acm.org/10.1145/1579114.1579146

[50] A. Dey, T. Sohn, S. Streng, and J. Kodama, "icap: Interactive prototyping of context-aware applications," in *Pervasive Computing*. Springer Berlin / Heidelberg, 2006, vol. 3968, pp. 254–271.

[51] K. Lyons, H. Brashear, T. Westeyn, J. S. Kim, and T. Starner, "Gart: the gesture and activity recognition toolkit," in *Proceedings of the 12th international conference on Human-computer interaction: intelligent multimodal interaction environments*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 718–727. [Online]. Available: http://dl.acm.org/citation.cfm?id=1769590.1769671

[52] D. Bannach, P. Lukowicz, and O. Amft, "Rapid prototyping of activity recognition applications," *Pervasive Computing, IEEE*, vol. 7, no. 2, pp. 22 –31, april-june 2008.

[53] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, and A. Brandle, "Rapid prototyping for pervasive applications," *IEEE Pervasive Computing*, vol. 6, no. 2, pp. 76 –84, april-june 2007.

[54] S. Greenberg and C. Fitchett, "Phidgets: easy development of physical interfaces through physical widgets," in *Proceedings of the 14th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2001, pp. 209–218.

[55] S. Tokuhisa, T. Ishizawa, Y. Niwa, K. Kasuya, A. Ueki, S. Hashimoto, K. Koriyama, and M. Inakage, "xtel: a development environment to support rapid prototyping of "ubiquitous content"," in *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*. New York, NY, USA: ACM, 2009, pp. 323–330.

[56] R. Ballagas, M. Ringel, M. Stone, and J. Borchers, "istuff: a physical user interface toolkit for ubiquitous computing environments," in *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2003, pp. 537–544.

[57] T. Matthews, A. K. Dey, J. Mankoff, S. Carter, and T. Rattenbury, “A toolkit for managing user attention in peripheral displays,” in *Proceedings of the 17th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2004, pp. 247–256. [Online]. Available: http://doi.acm.org/10.1145/1029632.1029676

[58] P. Sundström, A. Taylor, K. Grufberg, N. Wirström, J. Solsona Belenguer, and M. Lundén, “Inspirational bits: towards a shared understanding of the digital material,” in *Proceedings of the 2011 annual conference on Human factors in computing systems*. New York, NY, USA: ACM, 2011, pp. 1561–1570. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979170

[59] J. J. Barton and V. Vijayaraghavan, “Ubiwise, a ubiquitous wireless infrastructure simulation environment,” *HP Technical Reports*, 2002. [Online]. Available: http://home.comcast.net/~johnjbarton/ubicomp/ur/ubiwise/publications/ubiwise-05MAY02.pdf

[60] W. Jouve, J. Bruneau, and C. Consel, “Diasim: A parameterized simulator for pervasive computing applications,” in *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, 9-13 2009, pp. 1 –3.

[61] E. O’Neill, M. Klepal, D. Lewis, T. O’Donnell, D. O’Sullivan, and D. Pesch, “A testbed for evaluating human interaction with ubiquitous computing environments,” in *TRIDENTCOM ’05: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 60–69.

[62] J. C. Lee, D. Avrahami, S. E. Hudson, J. Forlizzi, P. H. Dietz, and D. Leigh, “The calder toolkit: wired and wireless components for rapidly prototyping interactive devices,” in *roceedings of the 5th conference on Designing interactive systems*. New York, NY, USA: ACM, 2004, pp. 167–175.

[63] L. Buechley, M. Eisenberg, J. Catchen, and A. Crockett, “The lilypad arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education,” in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2008, pp. 423–432. [Online]. Available: http://doi.acm.org/10.1145/1357054.1357123

[64] B. Hartmann, “Ex-a-sketch: Wizard of oz sketch animation for experience prototyping,” http://hci.stanford.edu/research/ex-a-sketch/.

[65] Modkit, http://www.modk.it/.

[66] T. S. McNerney, "From turtles to tangible programming bricks: explorations in physical language design," *Personal and Ubiquitous Computing*, vol. 8, pp. 326–337, 2004, 10.1007/s00779-004-0295-6. [Online]. Available: http://dx.doi.org/10.1007/s00779-004-0295-6

[67] M. S. Horn and R. J. K. Jacob, "Designing tangible programming languages for classroom use," in *Proceedings of the 1st international conference on Tangible and embedded interaction*. New York, NY, USA: ACM, 2007, pp. 159–162. [Online]. Available: http://doi.acm.org/10.1145/1226969.1227003

[68] L. Buechley and M. Eisenberg, "Boda blocks: a collaborative tool for exploring tangible three-dimensional cellular automata," in *Proceedings of the 8th iternational conference on Computer supported collaborative learning*. International Society of the Learning Sciences, 2007, pp. 102–104. [Online]. Available: http://dl.acm.org/citation.cfm?id=1599600.1599618

[69] B. Myers, S. Park, Y. Nakano, G. Mueller, and A. Ko, "How designers design and program interactive behaviors," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, sept. 2008, pp. 177 –184.

[70] S. Amershi, J. Fogarty, A. Kapoor, and D. Tan, "Overview based example selection in end user interactive concept learning," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 247–256.

[71] R. Fiebrink, P. R. Cook, and D. Trueman, "Human model evaluation in interactive supervised learning," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 147–156.

[72] S. Amershi, J. Fogarty, A. Kapoor, and D. S. Tan, "Effective end-user interaction with machine learning." in *AAAI*, 2011.

[73] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay, "Gestalt: integrated support for implementation and analysis in machine learning," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 2010, pp. 37–46.

[74] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1999, pp. 434–441.

[75] "Android sdk." [Online]. Available: http://developer.android.com/sdk/index.html

[76] "Adobe device central." [Online]. Available: http://www.adobe.com/products/creativesuite/devicecentral/features/

[77] Y. Nakanishi, K. Sekiguchi, T. Ohmori, S. kitahara, and D. Akatsuka, “Hybrid prototyping by using virtual and miniature simulation for designing spatial interactive information systems,” in *Pervasive Computing*, K. Lyons, J. Hightower, and E. Huang, Eds. Springer Berlin / Heidelberg, 2011, vol. 6696, pp. 250–257.

[78] J. Park, M. Moon, S. Hwang, and K. Yeom, “Cass: A context-aware simulation system for smart home,” *Software Engineering Research, Management and Applications, ACIS International Conference on*, vol. 0, pp. 461–467, 2007.

[79] H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto, and M. Ito, “Ubireal: Realistic smartspace simulator for systematic testing,” in *UbiComp 2006: Ubiquitous Computing*, P. Dourish and A. Friday, Eds. Springer Berlin / Heidelberg, 2006, vol. 4206, pp. 459–476.

[80] R. Morla and N. Davies, “Evaluating a location-based application: a hybrid test and simulation environment,” *Pervasive Computing, IEEE*, vol. 3, no. 3, pp. 48 – 56, july-sept. 2004.

[81] S. Dow, B. MacIntyre, J. Lee, C. Oezbek, J. Bolter, and M. Gandy, “Wizard of oz support throughout an iterative design process,” *Pervasive Computing, IEEE*, vol. 4, no. 4, pp. 18 – 26, oct.-dec. 2005.

[82] I. Culverhouse and S. Gill, “Bringing concepts to life: introducing a rapid interactive sketch modelling toolkit for industrial designers,” in *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*. New York, NY, USA: ACM, 2009, pp. 363–366. [Online]. Available: http://doi.acm.org/10.1145/1517664.1517737

[83] R. C. Davis, T. S. Saponas, M. Shilman, and J. A. Landay, “Sketchwizard: Wizard of oz prototyping of pen-based user interfaces,” in *Proceedings of the 20th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2007, pp. 119–128. [Online]. Available: http://doi.acm.org/10.1145/1294211.1294233

[84] T.-J. Nam, “Sketch-based rapid prototyping platform for hardware-software integrated interactive products,” in *CHI ’05 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2005, pp. 1689–1692. [Online]. Available: http://doi.acm.org/10.1145/1056808.1056998

[85] B. Hartmann, S. Follmer, A. Ricciardi, T. Cardenas, and S. R. Klemmer, “d.note: revising user interfaces through change tracking, annotations, and alternatives,” in *Proceedings of the 28th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2010, pp. 493–502. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753400

[86] G. Johnson, M. D. Gross, J. Hong, and E. Yi-Luen Do, “Computational support for sketching in design: A review,” *Found. Trends Hum.-Comput. Interact.*, vol. 2, pp. 1–93, January 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1576251.1576252

[87] Scratch for Arduino, http://seaside.citilab.eu/scratch/arduino. [Online]. Available: http://seaside.citilab.eu/scratch/arduino

[88] K. Reader, “Using storyboard techniques to identify design opportunities,” *The Technology Teacher*, April 2005.

[89] J. Forsyth and T. Martin, “Extracting behavioral information from electronic storyboards,” in *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2014, pp. 253–262.

[90] M. Brambilla, J. Cabot, and M. Wimmer, “Model-driven software engineering in practice,” *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[91] A. Van Deursen, P. Klint, and J. Visser, “Domain-specific languages: An annotated bibliography.” *Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[92] J.-P. Tolvanen and S. Kelly, “Defining domain-specific modeling languages to automate product derivation: Collected experiences,” in *Software Product Lines*. Springer, 2005, pp. 198–209.

[93] F. Hermans, M. Pinzger, and A. Van Deursen, “Domain-specific languages in practice: A user study on the success factors,” in *Model driven engineering languages and systems*. Springer, 2009, pp. 423–437.

[94] J. Whittle, J. Hutchinson, and M. Rouncefield, “The state of practice in model-driven engineering,” *Software, IEEE*, vol. 31, no. 3, pp. 79–85, 2014.

[95] G. Simko, D. Lindecker, T. Levendovszky, S. Neema, and J. Sztipanovits, “Specification of cyber-physical components with formal semantics–integration and composition,” in *Model-Driven Engineering Languages and Systems*. Springer, 2013, pp. 471–487.

[96] A. Gill, “Domain-specific languages and code synthesis using haskell,” *Queue*, vol. 12, no. 4, p. 30, 2014.

[97] MetaCase, “Metaedit+ modeler,” http://www.metacase.com/mep/.

[98] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel, “Design guidelines for domain specific languages,” *arXiv preprint arXiv:1409.2378*, 2014.

[99] O. M. Group, “Object constraint language 2.4,” http://www.omg.org/spec/OCL/2.4/.

[100] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. Polack, and G. Botterweck, "Taming emf and gmf using model transformation," in *Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 211–225.

[101] D. S. Kolovos, L. M. Rose, R. F. Paige, and F. A. Polack, "Raising the level of abstraction in the development of gmf-based graphical model editors," in *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*. IEEE Computer Society, 2009, pp. 13–19.

[102] "Cocovila," https://github.com/CoCoViLa/CoCoViLa.

[103] H. Cho, J. Gray, and E. Syriani, "Creating visual domain-specific modeling languages from end-user demonstration," in *Proceedings of the 4th International Workshop on Modeling in Software Engineering*. IEEE Press, 2012, pp. 22–28.

[104] L. Wouters, "Towards the notation-driven development of dsmls," in *Model-Driven Engineering Languages and Systems*. Springer, 2013, pp. 522–537.

[105] J. C. Grundy, J. Hosking, K. N. Li, N. M. Ali, J. Huh, and R. L. Li, "Generating domain-specific visual language tools from abstract visual specifications," *Software Engineering, IEEE Transactions on*, vol. 39, no. 4, pp. 487–515, 2013.

[106] D. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[107] A. G.-D. Dimitris Kolovos, Louis Rose and R. Paige, "The epsilon book," https://eclipse.org/epsilon/doc/book/, 2015.

[108] M. Famelis, S. Ben-David, M. Chechik, and R. Salay, "Partial models: A position paper," in *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*. ACM, 2011, p. 1.

[109] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, Nov. 2011.

[110] B. Buxton, "Remarks by bill buxton at mix09," http://news.microsoft.com/speeches/bill-buxton-mix09, 2009.

[111] C. Cassandras, *Introduction to Discrete Event Systems*. Springer, 2007.

[112] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky, "Toward patient safety in closed-loop medical device systems," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, New York, NY, USA, 2010, pp. 139–148.

[113] J. F. Allen, “Maintaining knowledge about temporal intervals,” *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983. [Online]. Available: http://doi.acm.org/10.1145/182.358434

[114] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime, “Uppaal-tiga: Time for playing games!” in *Computer Aided Verification*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4590, pp. 121–125.

[115] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, “Times: A tool for schedulability analysis and code generation of real-time systems,” in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, K. Larsen and P. Niebert, Eds. Springer Berlin Heidelberg, 2004, vol. 2791, pp. 60–72. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-40903-8_6

[116] Llus Mrquez and Xavier Carreras and Kenneth Litkowski and Suzanne Stevenson, “Semantic role labeling: An introduction to the special issue,” *Computational Linguistics*, vol. 34, June 2008.

[117] B. Y. Lim and A. K. Dey, “Toolkit to support intelligibility in context-aware applications,” in *Proceedings of the 12th ACM international conference on Ubiquitous computing*. New York, NY, USA: ACM, 2010, pp. 13–22. [Online]. Available: http://doi.acm.org/10.1145/1864349.1864353

[118] “Acceleo model to text,” http://www.eclipse.org/acceleo/, 2014.

[119] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, “Protothreads: simplifying event-driven programming of memory-constrained embedded systems,” in *Proceedings of the 4th international conference on Embedded networked sensor systems*, 2006, pp. 29–42.

[120] “Opensound control,” http://opensoundcontrol.org/, 2014.

[121] PlantUML, http://plantuml.sourceforge.net/, 2015.

[122] Arduino, www.arduino.cc.

[123] S. L. Star and J. R. Griesemer, “Institutional ecology,translations’ and boundary objects: Amateurs and professionals in berkeley’s museum of vertebrate zoology, 1907-39,” *Social studies of science*, vol. 19, no. 3, pp. 387–420, 1989.

[124] X. Zhou, M. Ackerman, and K. Zheng, “Cpoe workarounds, boundary objects, and assemblages,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3353–3362.

[125] W. G. Lutters and M. S. Ackerman, “Beyond boundary objects: collaborative reuse in aircraft technical support,” *Computer Supported Cooperative Work (CSCW)*, vol. 16, no. 3, pp. 341–372, 2007.

[126] M. Bohøj, N. G. Borchorst, N. O. Bouvin, S. Bødker, and P.-O. Zander, "Timeline collaboration," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 523–532.

[127] C. Lee, "Boundary negotiating artifacts: Unbinding the routine of boundary objects and embracing chaos in collaborative work," *Computer Supported Cooperative Work*, vol. 16, no. 3, pp. 307–339, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10606-007-9044-5

[128] M. K. Halpern, I. Erickson, L. Forlano, and G. K. Gay, "Designing collaboration: Comparing cases exploring cultural probes as boundary-negotiating objects," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, 2013, pp. 1093–1102.

[129] P. Dalsgaard, K. Halskov, and D. A. Basballe, "Emergent boundary objects and boundary zones in collaborative design research projects," in *Proceedings of the 2014 conference on Designing interactive systems*. ACM, 2014, pp. 745–754.

[130] M. K. Halpern, "Across the great divide: Boundaries and boundary objects in art and science," *Public Understanding of Science*, p. 0963662510394040, 2011.

[131] F. Gemperle, C. Kasabach, J. Stivoric, M. Bauer, and R. Martin, "Design for wearability," in *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*. IEEE, 1998, pp. 116–122.

[132] S. L. Star, "This is not a boundary object: Reflections on the origin of a concept," *Science, Technology & Human Values*, vol. 35, no. 5, pp. 601–617, 2010.

[133] A. Fong, R. Valerdi, and J. Srinivasan, "Boundary objects as a framework to understand the role of systems integrators," in *Systems Research Forum*, vol. 2, no. 01. World Scientific, 2007, pp. 11–18.

[134] P. Carlile, "A pragmatic view of knowledge and boundaries: Boundary objects in new product development," *Organizational Science*, vol. 13, no. 4, pp. 442–455, July - August 2002.

[135] D. Nicolini, J. Mengis, and J. Swan, "Understanding the role of objects in cross-disciplinary collaboration," *Organization Science*, vol. 23, no. 3, pp. 612–629, 2012.

[136] G. Fischer, "Social creativity: turning barriers into opportunities for collaborative design," in *Proceedings of the eighth conference on Participatory design: Artful integration: interweaving media, materials and practices-Volume 1*. ACM, 2004, pp. 152–161.

[137] M. Tohidi, W. Buxton, R. Baecker, and A. Sellen, "Getting the right design and the design right," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006, pp. 1243–1252.

[138] G. Fischer, “Distances and diversity: sources for social creativity,” in *Proceedings of the 5th conference on Creativity & cognition*. ACM, 2005, pp. 128–136.

[139] B. E. John, L. Bass, R. Kazman, and E. Chen, “Identifying gaps between hci, software engineering, and design, and boundary objects to bridge them,” in *CHI ’04 Extended Abstracts on Human Factors in Computing Systems*, 2004, pp. 1723–1724.

[140] A. Walenstein, “Finding boundary objects in se and hci: An approach through engineering-oriented design theories.” in *ICSE Workshop on SE-HCI*. Citeseer, 2003, pp. 92–99.

[141] A. F. Phelps and M. Reddy, “The influence of boundary objects on group collaboration in construction project teams,” in *Proceedings of the ACM 2009 International Conference on Supporting Group Work*, 2009, pp. 125–128.

[142] L. A. Dabbish, P. Wagstrom, A. Sarma, and J. D. Herbsleb, “Coordination in innovative design and engineering: observations from a lunar robotics project,” in *Proceedings of the 16th ACM international conference on Supporting group work*. ACM, 2010, pp. 225–234.

[143] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson, “How does radical collocation help a team succeed?” in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 2000, pp. 339–346.

[144] P. Heinrich, M. Kilic, F.-R. Aschoff, and G. Schwabe, “Enabling relationship building in tabletop-supported advisory settings,” in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 171–183.

[145] F. E. Ritter, G. D. Baxter, and E. F. Churchill, *Foundations for Designing User-Centered Systems*. Springer, 2014.

[146] J. Brooke, “Sus-a quick and dirty usability scale,” *Usability evaluation in industry*, vol. 189, p. 194, 1996.

[147] W. Albert and T. Tullis, *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013.

[148] J. Lee, L. Garduño, E. Walker, and W. Burleson, “A tangible programming tool for creation of context-aware applications,” in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013, pp. 391–400.

[149] M. T. Blake, “An ambulatory monitoring algorithm to unify diverse e-textile garments,” Master’s thesis, Virginia Tech, 2014.

[150] E. Y.-L. Do, “Design sketches and sketch design tools,” *Knowledge-Based Systems*, vol. 18, no. 8, pp. 383–405, 2005.

[151] Y. Oh, M. D. Gross, S. Ishizaki, and E. Y.-L. Do, “A constraint-based furniture design critic,” *Research and Practice in Technology Enhanced Learning*, vol. 5, no. 02, pp. 97–122, 2010.

[152] A. Payne and J. K. Johnson, “Firefly Experiments,” http://fireflyexperiments.com/.

[153] B. Hartmann, “Gaining design insight through interaction prototyping tools,” Ph.D. dissertation, Stanford University, 2009.

[154] Balsamiq, http://www.balsamiq.com/products/mockups.

[155] CogSketch, http://www.qrg.northwestern.edu/software/cogsketch/, 2015.