Extracting Behavioral Information from Electronic Storyboards

Jason Forsyth
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
jforsyth@vt.edu

Tom Martin
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
tlmartin@vt.edu

ABSTRACT

In this paper we outline methods for extracting behavioral descriptions of interactive prototypes from electronic storyboards. This information is used to help interdisciplinary design teams evaluate potential ideas early in the design process. Using electronic storyboards provides a common descriptive medium where team members from different disciplinary backgrounds can collectively express the intended behavior of their prototype. The behavioral information is extracted by a combination of visual tags applied to elements of the storyboard, analysis of storyboard layout, and natural language processing of text written in the frames. We describe this process, provide a proof of concept example, and discuss design choices in developing this tool.

Author Keywords

Prototyping; Interdisciplinary Design; Programming Tools

ACM Classification Keywords

D.2.2 Design Tools and Techniques

INTRODUCTION

In this paper we present a novel information extraction process to create behavioral models of interactive systems from electronic storyboards. Our approach allows storyboards to serve as a common design tool for interdisciplinary teams and enable those teams to reason about system behavior or aid in implementation. We focus on the early stages of the design cycle where ideas are often in flux. As such, the artifacts generated by this process are not final products, but rough examples that are useful for generating discussion amongst the design team. This type of prototyping is important during the early stages of design when many ideas regarding form, function, and behavior are being evaluated [28].

EICS 2014, June 17–20, 2014, Rome, Italy.

http://dx.doi.org/10.1145/2607023.2607034

In an interdisciplinary setting, the insight and experience of all team members is required, especially when creating and evaluating prototypes. The close proximity of technology and human experience, inherent in pervasive computing systems, necessitates that these systems be designed by the domain experts in the fields in which the systems will be deployed. It is not the engineers, who until now are the typical creators of pervasive systems, but the fashion designers, industrial designers, and architects that are trained to work in these domains. However, designing pervasive systems requires knowledge of computing technologies and programming languages that may be inaccessible to team members without a computing background. Without appropriate design tools and methods, the insights of non-computing team members may be lost.

To address these problems, many tools have been created that attempt to lower the barrier to learning programming languages [4, 21, 26], or are customized tools that target domain specific applications [16, 18, 24, 29]. However, these tools present their own difficulties. First, even a simplified programming language can be a barrier for team members who are not programmers. For these members, their understanding of the prototype's behavior is solely dependent upon how well they understand the programming language. Second, domain specific tools can be suitable for specific projects, but they may be cumbersome for a general design process where the target application can change between projects. Thus, a team may have a suitable tool for describing wearable applications, but then be forced to use a different tool for location-based systems.

We propose the use of electronic storyboards to enable interdisciplinary teams to collaborate in the designing and prototyping of pervasive computing systems. Electronic storyboards are a software design tool that allow an interdisciplinary team to electronically draw and depict how a user, or group of users, interacts with some pervasive computing system. Storyboards are advantageous because they can represent high-level behavior such as context, location, action, and temporal phenomena [12] [9, p.296] in multiple application domains and are a format in which team members can collectively reason about system behavior [14]. We intend for these sto-

ryboards to be employed during the early design stages when quick, low-fidelity prototypes are desirable.

Our work seeks to extend storyboards from a descriptive medium, where ideas are illustrated, to one that is generative and can be used to prototype design ideas. We achieve this by extracting behavioral information from storyboards to form a model of computation that describes the depicted prototype. This model can be used by the design team to reason about the prototype's behavior or it can be synthesized into source code using traditional model-driven engineering techniques [3, 8]. The ability to automatically create behavioral models from the storyboard reduces the time between design iterations and allows the team to produce more prototypes over time, resulting in better design outcomes [22]. Furthermore, the process of creating the behavioral model is interactive and may reveal missing or ambiguous information in the storyboard thereby facilitating discussion of intended behaviors.

In this paper, we address the key difficulty of extracting a suitable model of computation from electronic storyboards. We present an information extraction method where the team as a whole can collaborate on designing prototypes by "tagging" visual elements of the storyboard that represent specific computing domain concepts such as Event, Action, Time, and Context. Based upon the user tags, the layout of the storyboard, and the natural language processing of textual annotations, a timed automaton model of the prototype can be formed.

The remainder of this paper presents our process for creating the timed automaton and discusses design tradeoffs for this approach. The following sections present related work on prototyping tools and motivates the need for interdisciplinary design of pervasive computing systems. Next, we present an overview of the design challenges in extracting behavioral information from the electronic storyboards as well as a description of our approach. Finally, we provide a proof-of-concept example that describes how a timed automaton can be formed. While only a singular example, it showcases many common problems faced when extracting information and is representative of a wider class of problems.

BACKGROUND AND RELATED WORK

This section outlines the need for interdisciplinary design of pervasive computing systems, describes the drawbacks of existing prototyping tools, and discusses related work in storyboarding.

Need for Interdisciplinary Design

The goal of pervasive computing is to make interactions with computing technology subconscious, or cognitively and physically invisible. Weiser said that our computers should be "an invisible foundation that is quickly forgotten but always with us, and effortlessly used throughout our lives" [32]. Achieving these "invisible" interactions requires detailed knowledge about the needs and desires of the end user. To understand these needs, creators of pervasive systems must "uncover the very practices through which people live and to make these invisible practices visible and available to the developers of ubicomp environments" [1]. Understanding these invisible interactions requires an interdisciplinary approach that includes not just technology researchers and engineers, but also designers, particularly but not limited to industrial design, architecture, and apparel. It is these practitioners who are trained to understand human behavior and the subconscious motivations and interactions of the end user [31, 32]. Working in an interdisciplinary setting aligns with the foundations and goals of pervasive computing by combining a revolution in technology with a focus on human experience that makes human-computer interaction calming and supportive.

Existing Tools and Related Work

Creating meaningful pervasive computing interactions is a balance of constraints between the user experience, physical form of the system, and the underlying computational platform. In an interdisciplinary setting, the introduction of computing technology, such as microcontrollers, sensors, and actuators, may exclude team members from fully engaging in the design process as they are unfamiliar with particular technologies, or have a limited understanding of programming concepts. While it is not expected that every team member should be an expert programmer, or like-wise an expert designer, each member should have sufficient knowledge and tools to work across disciplines.

Several tools have been created that address these problems and fall into two large categories: either general programming languages that are aimed at novices, such as Scratch For Arduino [26] and ModKit [6], or custom tools targeted at a particular application domain, such as dTools [16], aCAPpella [11], CAMP [29], or Activity Designer [18].

In an interdisciplinary setting either of these approaches may still create difficulties. Requiring non-programmers to learn a programming language creates an additional barrier for those members to engage in prototyping. Depending on the level of abstraction in the programming language, team members may be caught up in the details of the underlying implementation, rather than reasoning about why a behavior did not work. As we will describe in later sections, our electronic storyboards address these issues by providing a high level of abstraction that describe systems in terms of State, Event, Action, and Context.

A similar difficulty is encountered by domain-specific tools as they are tailored to work well in a particular domain but may not be usable in all situations. Considering that pervasive computing applications span a wide range from wearable computers to ambient spaces, it is undesirable for design teams to switch tools when working on different projects. Electronic storyboards address this issue by utilizing the storyboard format that has been used to express applications in domains such as

augmented reality [27], conditional events for location-based applications [33], mixed-media applications [7], or creating GUI applications [17, 23, 25]. Additional work, such as the MuiCSers framework and Timisto have focused on using storyboards as a general design tool by extracting task sequences, abstract user interfaces, and time sequences from storyboards [15, 30].

While electronic storyboards address several issues, they are not a replacement for all domain-specific tools or programming languages. For certain applications it may not be suitable to use a storyboard to describe the application. Furthermore, because our approach describes applications at a high-level of abstraction, certain domain-specific concepts may be difficult to implement or express. However, our electronic storyboard could still be useful as the behavioral models generated from our tool could serve as an input to existing tools.

MOTIVATION AND CHALLENGES USING ELECTRONIC STORYBOARDS

In the previous section, we discussed the need for interdisciplinary design of pervasive computing systems and the difficulties in existing design tools. To address these problems we advocate the use of electronic storyboards for interdisciplinary teams. Here we outline the requirements needed to extract behavioral information from electronic storyboards and how to map that information to a suitable model of computation. We intend these storyboards to be used across application domains, and to be employed during the early design stages when quick, low-fidelity prototypes are desirable.

In the remainder of this section we provide establishing definitions and discuss the design tradeoffs when using electronic storyboards. Specifically, we address the (1) tension between the ambiguous nature of storyboards, and the specificity of software systems, (2) how prototype complexity is represented in the storyboard, and (3) the need for storyboards to represent key semantics such as time, action, and context, and how that information maps to a model of computation.

Definitions and Assumptions

Electronic storyboards are a software design tool that allow 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. 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 paper 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.

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 1: Supported tags for storyboard objects

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 [9, 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 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) [10] 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 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, 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 that are read left to right. 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 1a. However, when developing interactive systems, there are often conditional or iterative behaviors that need to be expressed. For example, a device should make a choice between two inputs, or should continue a behavior until some condition is met.

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 1b. In this way, the linear structure can be extended to exhibit branching behavior. These arrows can also be used to express looping behavior where the storyboard returns back on itself as in Figure 1c. 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 aid in generating 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 code generation to enable rapid creation of the 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. 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 [5]. Additionally, several of the keywords in Table 1 map directly to timed automaton concepts as shown in Figure 2. 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 [2] that is used to specify how those automata should be ordered and executed. Finally, timed automata can be used to automatically generate code [3, 8] that can facilitate implementing the prototype once the model has been formed.

INFORMATION EXTRACTION METHOD

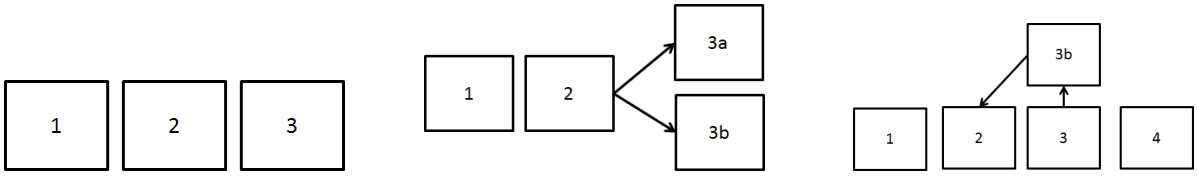
In this section, we discuss our approach for extracting behavioral information from an electronic storyboard. We describe how storyboard information, expressed through a series of keyword tags and natural language processing of textual information can be transformed into a timed automaton. Figure 2 provides a graphical description of how storyboard information can be mapped to the timed automaton. As we will discuss later in this section, images that have been tagged in the storyboard can directly map to timed automaton elements, while textual information must be parsed before use.

The process described in this section has been implemented using the Eclipse Graphical Editor Framework [13] which allows users to electronically draw and annotate a storyboard. A screenshot of the GUI as seen by the user is shown in Figure 3. The remainder of this section describes the specifics of how the storyboards are implemented in GEF, and how the storyboard model is transformed into a timed automaton.

Electronic Storyboards in Eclipse GEF

An implementation of electronic storyboards has been created in GEF that allows users to draw a storyboard using frames, images, and text. Images placed on the storyboard canvas can then be "tagged" using the set of keywords in Table 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. 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



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

Figure 1: 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.

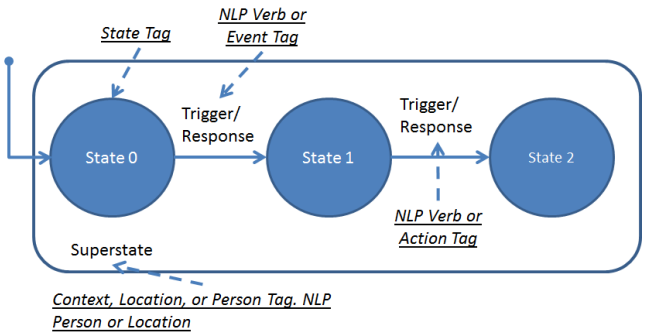


Figure 2: Mapping from storyboard objects onto a timed automaton

can be modified in the Properties View below the canvas. In Figure 3, 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.

Transforming Storyboards into Timed Automata

In the previous sections, we discussed how electronic storyboards are implemented in Eclipse. This section presents a method for converting electronic storyboard information into timed automata. The first subsection describes how storyboard information corresponds to timed automaton objects. The timed automaton can then be synthesized in two phases: first, the layout of the storyboard is partitioned into regions of similar time and context then each region is compiled into an automaton.

Mapping Storyboard Objects to Timed Automata

Within electronic storyboards, there are two main types of information: visual images that have been tagged by the design team and the results of natural language processing from any text or words placed on the storyboard. Figure 2 shows how these two sets of information can be mapped to a timed automaton. Using the keywords in Table 1 tagged visual annotations within the storyboard can be directly mapped to timed automaton elements, e.g. State, Action, Event, and Context. For example, an Event tag corresponds with a state transition trigger, while an Action tag maps to a responsive action by the automaton.

Textual annotation are parsed using Semantic Role Labeling (SRL) and Named Entity Recognition (NER) using the Senna Natural Language Processing (NLP) tool [10]. SRL identifies the structure and content of a sentence, such as verbs, direct object, indirect object, temporal modifiers, or locations. NER identifies words that are a person, location, or time indicator. These natural language results aid in capturing behavioral information that is not explicitly tagged by the design team. For example, if an Action or Event is not tagged, its existence could be inferred by a verb clause found by SRL.

Because of the ambiguity in natural language processing results, this information is considered less authoritative and does not automatically map to timed automaton concepts. When parsing the storyboard, if certain tagged information is missing, the NLP results are consulted for the missing information. However, as we will see in the following section, the user is queried before any NLP information is used to create the timed automaton.

Partition Based Upon Time and Context

The first step in creating the timed automaton is to partition the storyboard into regions of similar context and time. At present, partitioning is done by "reading" the storyboard frame by frame. For a group of frames, the first context or time interval encountered is set as the initial value. If new contexts or time intervals are encountered, the previously read frames are partitioned into their own set and a new set is created with the newly encountered time or context.

Context and time information can be found from Context, Temporal, or Location tags, or from location results from natural language processing. Currently, the tool can identify that some information is related to con-

text 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 [11], or existing architectures to recognize and disseminate context [19]. 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 [2] to provide a formal definition of temporal ordering.

Building Local Behavior

After the storyboard has been partitioned into sets of similar time and context, each set is analyzed to build a timed automaton. The storyboard is read according to its layout and each frame is analyzed for information relating to the behavior of the prototype. The tool looks for State, Event, and Action tagged images which correspond to states of the automaton, the triggering events between states, and the actions taken by the system. Parsing continues until one of two stop conditions is reached: two states have been found, or an event and action have been found. Each condition reflects that a state transition has occurred indicating the depicted prototype executed some behavior. With two states, it is known that 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. In the absence of tagged objects, the textual annotations within the storyboard are queried based upon SRL and NER parsing results. If NLP information 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.

FEASIBILITY OF EXTRACTING INFORMATION FROM A 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, is taken from an interdisciplinary product design course [20] 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, it 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 "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 paper 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 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, 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 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 4 is generated by the process. We illustrate how that automaton is created in the remainder of this section.

Partitioning Based Upon Time and Context

Beginning with the first frame, the storyboarding tool searches for context and time information. Initially the

Figure 3: 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.

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 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 4 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, 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

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

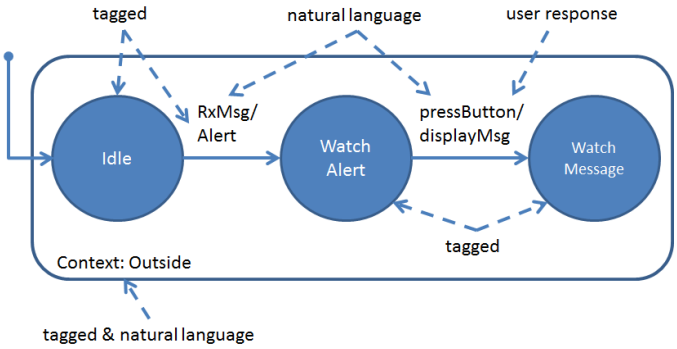


Figure 4: Timed automaton derived from example storyboard. Information sources are indicated with dashed arrows.

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 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 4 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 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 4 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 4 represents the behavioral model produced by this process. Additionally, source code can be generated from the automaton. Program 1 shows a portion of generated code that shows the transition logic for the automaton using the Arduino programming language [4]. The code initially checks to if "outside" is the current context and then executes the logic defined by the automaton.

DISCUSSION

In the previous section we showed the feasibility of extracting behavioral models from an electronic storyboard. While the process of deriving the model can seem involved, from the perspective of the user, creating the timed automaton takes less than a minute. The time required to create the automaton is dependent upon the information content of the storyboard. If all the necessary States, Events, and Actions are already tagged, then producing the automaton is automatic. However, if information is missing, the user will be queried until the information is found, which may be cumbersome if there are many NLP results.

While the example storyboard could be complied into an automaton, not all storyboards can be. In particular, the tool has difficulty identifying State information that is not tagged. Events and Actions can be easily found from verbs in natural language processing, but State information does not have a natural analogue. Without tagged States, the tool cannot separate events and actions into individual transitions. The primacy of state

```
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;
}
```

Program 1: Arduino code created from the timed automaton in Figure 4

information requires that when using the tool teams take care to tag States and may need specific instruction to do so.

Finally, one drawback of our electronic storyboarding approach is that generated models and code are general and do not represent a specific domain. While this allows design teams to describe applications across domains, our approach does not facilitate automatic implementation of a fully functioning prototype. For the code shown in Program 1, the design team would need to implement the functionality to check for a message, beep, or display the message. Given that the time required to synthesize the storyboard and generate the code is short, our approach can reduce the time between prototypes by automatically implementing the logical structure of the prototype.

CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a process for automatically extracting behavioral information from electronic storyboards of pervasive computing systems. We have validated this approach by implementing a proof-of-concept tool that parses an electronic storyboard, generates a timed automaton of the storyboard, and automatically synthesizes code for the prototype. We have pursued the use of electronic storyboards because we believe automatic synthesis of storyboards will reduce the time between prototypes and increase the number of prototypes that can be created by an interdisciplinary team, thereby improving their ability to explore the design space.

For future work, we plan to evaluate our storyboarding tool with novice and expert users to assess how the tool impacts their design processes. We also plan to improve the tool's capabilities by providing shared views of the design from various perspectives. It would also be useful to have changes in the timed automaton cause the storyboard to change, so that a team can see how changes in the model affect the storyboard or vice versa. Finally, we would like to integrate physical CAD tools (e.g., 3D renderings) with the storyboard tool so that a change in the form factor of the design would propagate to the storyboard.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their helpful comments on this work. This material is based upon work supported by National Science Foundation under Grant No. EEC-0935103 and the Virginia Tech Institute for Creativity Arts and Technology.

REFERENCES

1. Abowd, G., Mynatt, E., and Rodden, T. The human experience [of ubiquitous computing]. *IEEE Pervasive Computing 1*, 1 (Jan 2002), 48–57.
2. Allen, J. F. Maintaining knowledge about temporal intervals. *Commun. ACM 26*, 11 (Nov. 1983), 832–843.
3. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., and Yi, W. Times: A tool for schedulability analysis and code generation of real-time systems. In *Formal Modeling and Analysis of Timed Systems*, K. Larsen and P. Niebert, Eds., vol. 2791 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, 60–72.
4. Arduino. www.arduino.cc.
5. Arney, D., Pajic, M., Goldman, J. M., Lee, I., Mangharam, R., and Sokolsky, O. 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), 139–148.
6. Baafi, E., and Millner, A. A toolkit for tinkering with tangibles and connecting communities. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, ACM (New York, NY, USA, 2011), 349–352.
7. Bailey, B. P., Konstan, J. A., and Carlis, J. V. DEMAIS: designing multimedia applications with interactive storyboards. In *Proceedings of the ninth ACM international conference on Multimedia*, ACM (New York, NY, USA, 2001), 241–250.
8. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., and Lime, D. Uppaal-tiga: Time for playing games! In *Computer Aided Verification*, vol. 4590 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, 121–125.

9. Buxton, B. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, 2007.
10. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. *Journal of Machine Learning Research 12* (Nov. 2011), 2493–2537.
11. Dey, A. K., Hamid, R., Beckmann, C., Li, I., and Hsu, D. a cappella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (New York, NY, USA, 2004), 33–40.
12. Dow, S., Saponas, T. S., Li, Y., and Landay, J. A. External representations in ubiquitous computing design and the implications for design tools. In *Proceedings of the 6th conference on Designing Interactive Systems* (June 2006), 241–250.
13. Eclipse Graphical Editor Framework. http://www.eclipse.org/gef/.
14. Haensen, M. *User-Centered Process Framework and Techniques to Support the Realization of Interactive Systems by Multi-Disciplinary Teams*. PhD thesis, Universiteit Hasselt, 2011.
15. Haesen, M., Coninx, K., Bergh, J., and Luyten, K. Muicser: A process framework for multi-disciplinary user-centred software engineering processes. In *2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams*, Springer-Verlag (Berlin, Heidelberg, 2008), 150–165.
16. Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, ACM (New York, NY, USA, 2006), 299–308.
17. Li, Y., Cao, X., Everitt, K., Dixon, M., and Landay, J. A. Framewire: a tool for automatically extracting interaction logic from paper prototyping tests. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, ACM (New York, NY, USA, 2010), 503–512.
18. Li, Y., and Landay, J. A. 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), 1303–1312.
19. Lim, B. Y., and Dey, A. K. Toolkit to support intelligibility in context-aware applications. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, ACM (New York, NY, USA, 2010), 13–22.
20. Martin, T., Kim, K., Forsyth, J., McNair, L., Coupey, E., and Dorsa, E. Discipline-based instruction to promote interdisciplinary design of wearable and pervasive computing products. *Personal and Ubiquitous Computing* (December 2011), 1–14.
21. Modkit. http://www.modk.it/.
22. Moggridge, B. *Designing Interactions*. MIT Press, 2007.
23. Nam, T.-J. Sketch-based rapid prototyping platform for hardware-software integrated interactive products. In *CHI '05 extended abstracts on Human factors in computing systems*, ACM (New York, NY, USA, 2005), 1689–1692.
24. Obrenovic, Željko and Martens, Jean-Bernard. Sketching interactive systems with sketchify. *ACM Trans. Comput.-Hum. Interact. 18* (May 2011), 4:1–4:38.
25. Schmieder, P., Plimmer, B., and Vanderdonckt, J. Generating systems from multiple sketched models. *Journal of Visual Languages & Computing 21*, 2 (2010), 98 – 108.
26. Scratch for Arduino. http://seaside.citilab.eu/scratch/arduino.
27. Shin, M., soo Kim, B., and Park, J. Ar storyboard: An augmented reality based interactive storyboard authoring tool. *IEEE / ACM International Symposium on Mixed and Augmented Reality 0* (2005), 198–199.
28. Tohidi, M., Buxton, W., Baecker, R., and Sellen, A. Getting the right design and the design right. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, ACM (New York, NY, USA, 2006), 1243–1252.
29. Truong, K., Huang, E., and Abowd, G. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *Proceedings of Ubicomp 2004* (2004), 143–160.
30. Vogt, J., Haesen, M., Luyten, K., Coninx, K., and Meier, A. Timisto: A technique to extract usage sequences from storyboards. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '13, ACM (New York, NY, USA, 2013), 113–118.
31. Weiser, M. Some computer science issues in ubiquitous computing. *Communications of the ACM 36* (July 1993), 75–84.
32. Weiser, M. The world is not a desktop. *Interactions 1*, 1 (Jan. 1994), 7–8.
33. Welbourne, E., Balazinska, M., Borriello, G., and Fogarty, J. Specification and verification of complex location events with panoramic. In *Pervasive Computing*, vol. 6030 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, 57–75.