Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Bachelorarbeit

Studienfach
Informatik

Felix Kampfer

Situation-Aware User Interface Optimization

Prüfer: Prof. Dr. Christian Märtin
Zweitprüfer: Prof. Dr. Hubert Högl
Abgabe der Arbeit am: 20.03.2018

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Verfasser der Abschlussarbeit:
Felix Kampfer
Baumkirchner Str. 15
81673 München
Telefon:+49 152 04839964
felix.kampfer@gmail.com

Hochschule
Augsburg University of
Applied Sciences

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

_____ _____

Ort, Datum Unterschrift des/der Studierenden

Contents

Abstract

Despite decades of progress in user experience design, the vast majority of user interfaces fail to take into account a user's emotional and physical state, thereby severely limiting the kinds of interactions that can occur between man and machine, in addition to impeding existing processes that depend on user responses. The purpose of this thesis is to broaden the playing field of human-machine interaction by introducing a generalized situation-aware user interface optimization system.

We present a system that can analyze a user's physiological data to optimize UI elements to match his or her current center of attention, emotional state, level of excitement, and application context. Using our rule editor, application administrators can define their own adaptation rules with conditions and actions that are evaluated continuously. These situations can include recognizing a user's lowered pulse in a web application, the awareness of positive surprise in an e-commerce application, or detecting aggravation while driving a car. Application designers can use our system to process unfiltered user feedback and adapt interfaces to match any user's needs, personal interests or emotional state in a given context. We hope that increasing the accessibility and situational awareness of computers in this manner will usher in an era of more intuitive user interfaces.

1 Introduction

This section introduces the reasoning and goals of the project, in addition to describing the structure of this thesis.

1.1 Motivation

With an ever-increasing number of internet users [1], e-commerce customers [2], and smartphone users worldwide [3], good interface design is more important than ever.

User interfaces are strongly influenced by three factors: The capabilities and expectations of the user, the limitations of the medium upon which they are rendered, and the tools available to the developer. In the late 1970s and early 1980s, computers with window-based GUIs were being made available to a wide audience of consumers for the first time, as exemplified by Apple's Lisa [4]. The transition from a text-based command line interface to a graphical UI vastly improved the accessibility of computers and acted as a boon to the average citizen's ability to operate a computer.

Thirty years later, it was again one of Apple's flagship products that marked the start of a new paradigm in interface design: The Apple iPhone. The device introduced an immensely successful product series centered around a touchscreen, a technological medium that presented many new opportunities for application developers. It is now the task of these developers to revolutionize the industry once more, to make computers more accessible, and to improve the lives of everyone who uses software.

1.2 Goal

The aim of this paper is to contribute to the advancement of tools available to application developers and UI designers. More specifically, a semi-automated UI optimization engine with access to a user's vital signs, physiological data and derived emotional state is introduced. We hope that developers can take advantage of this feedback-based tool to create applications that can react to a user's raw emotions, possibly allowing users to accomplish tasks in a more natural and efficient manner, and, in the future, removing sources of frustration from modern web applications altogether.

This thesis will document the steps taken to develop a new version of the SitAdapt system. While the predecessor SitAdapt 1.0 [5] showcased the basics of how physiological data can be used to change a web application's appearance in real-time, the new system ("SitAdapt 2.0") features the ability to respond to an arbitrary selection of situations with a customized set of actions, and includes an additional input source.

1.3 Structure of this Thesis

This thesis is split into two main parts; a background section outlining the theoretical and practical foundations of the system, and an implementation section offering an overview of the system's components. Later sections evaluate the system and describe possible areas of improvement.

Section 2 presents the theoretical background of the system and and takes a look at modern user experience design philosophy, at the basics of detecting situations in a real-time monitoring system, at the difference between model- and pattern-based user interface design, at the various sensors and why we use them, at utilized web technologies, at our server and database framework selection, at the foundations of the SitAdapt system (including the original prototype, SitAdapt 1.0), and at other similar approaches that implement a reactive system in conjunction with a data processing pipeline.

Section 3 describes the implementation details of SitAdapt 2.0. It provides an overview of the system's architecture and then explains the system's individual components in detail, specifically the central cross-component communication module (the Database Writer), the external condition and situation configuration tool (the Rule Editor), the hardware sensor subscription and processing module (the Recording Component), and the UI change execution module (the EvaluationAndAdaptation Component). Next, the selection and implementation of adaptation options for a frontend web application are discussed. Finally, a walkthrough of the system is presented.

Section 4 summmarizes the accomplishments of the project and describes ethical considerations for the system's use.

Section 5 describes potential areas of improvement for the system in terms of distribution, feature enhancement, UI adaptation and software development.

The appendix (Section 7) includes descriptions of data received from the hardware sensors, the customer requirements specification, and instructions (in German) for installing, configuring and running the system.

2 Background

2.1 Theory

2.1.1 User Experience Design

Human-Computer Interaction (HCI) researchers in the 1970s and 80s were still looking for a name for their field when they described theories in the domains of ”artificial psycholinguistics”, ”cognitive ergonomics”, and ”software psychology”. [6] Now labeled *user experience design*, the field encompasses a variety of disciplines, spanning from the psychology of color and behavior to the technical details of interaction hardware and graphical user interface elements.

Terminology Used in This Thesis

To start with, the terminology used throughout this thesis will be clarified. Figure 1 shows how the concepts overlap with one another.

Usability refers to the state of being convenient and practicable for use [7]. Usability engineering (UE) describes the process of bringing about this quality, which includes setting up hypotheses, testing them, and revising features in a continuous process focused on achieving the goals of ”learnability, efficiency, memorability, error-free use, and subjective satisfaction” for a given product. [8]

Human-computer interaction (commonly abbreviated HCI) is the overarching topic of designing hardware and/or software-based intermediaries to facilitate translating a user’s intentions to a machine and providing output. It concerns itself with organisational, environmental, ergonomic, task- and constraint-based issues, in addition to system integration matters [9]. HCI research can focus on the construction of interfaces via experimental pursuits (e.g. new software infrastructure and mobile hardware devices) and on developing new interactive systems.

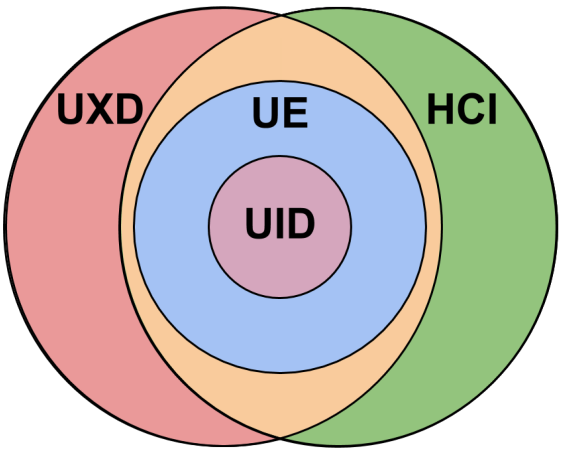

Figure 1: An overview of the related concepts

User experience design (UXD) places ”an emphasis on the inclusion of all customer processes that lead to and follow the usage of a product in the design process.” [10] It focuses on crafting a product’s ”process”, meaning the planning, implementation, and iterative improvement of a customer’s complete interaction with a product. The field encompasses various disciplines, including design (information design, service design, visual design, and industrial design), communication (content

management and branding), and optimization research (stakeholder research, human factor considerations, and usability). [10]

User experience design involves similar topics as HCI research, but stems from a more practical, business-oriented approach. [11] According to [12], HCI researchers, tied to academia, ”have the luxury of failing frequently, but also innovating rapidly, working under fewer constraints”, user experience design instead revolves around the limited requirements of industry professionals, who are, ”for example, working on tightly defined projects with tight budgets and timelines; needing to satisfy clients and stakeholders and mesh with people in other disciplines” [12].

User Interface Design (UID) is a subset of user experience design, limited to the design and implementation of what the user sees on a screen, namely graphical user interfaces (GUIs). All interfaces are made up of many layers, ranging from the pixels drawn on-screen by the graphics card, the operating system active on the user's machine, and the currently running browser application all the way to currently displayed website's internal architecture comprised of dynamically-loading content, interactive elements and pre-processed styling information. This thesis will focus primarily on web applications, although the UI optimization possibilities of the presented system are largely applicable to a variety of contexts and environments.

Technological Development: Form Follows Function

Interface design theory has always been tied to the capabilities and historical background of interface technology. The following overview of the evolution of hardware will seek to provide an understanding of the origins of today's software-focused advancements.

In 1822, the inventor Charles Babbage designed and partially developed a mechanical device (”the Difference Engine”) to automate the process of calculating mathematical expressions, featuring multiple ten-tooth-gears that could perform a single calculation on large numbers. Mechanical advancements in the later 19th century made it increasingly feasible to construct more elaborate machines, as evidenced by the first commercially available typewriter in 1874 (of Remington Arms Co.) and a census tabulating machine in 1884 (of Herman Hollerith). [13]

Whereas communications equipment of the 1930s relied on electro-mechanical relays, the ENIAC (Electronic Numerical Integrator and Calculator) of the 1940s was the first computer to feature vacuum tubes. The machine's 18000 vacuum tubes occasionally blew out, however. [14]. A solution to this problem was found in transistors, which needed much less power, space and time to work. They were first found in Bell Labs' TRADIC (for TRAnsistorized DIgital Computer) of 1954.[15].

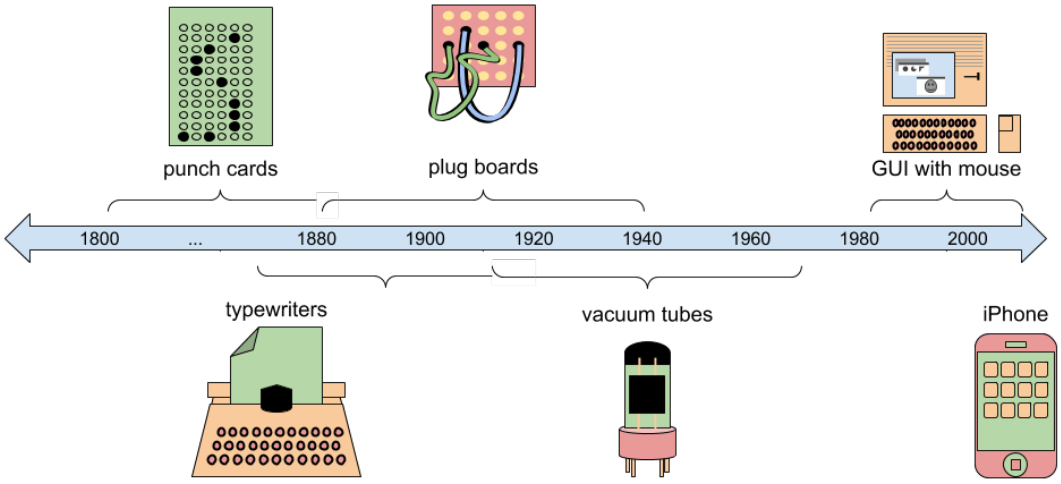

Figure 2: A simplified overview of interface hardware evolution

It took until the 1970s for computers to evolve from using a ticker tape as output. By then, a pattern of flashing lights on the front panel might indicate the results of the panel (more specifically, the contents of its registers), as was the case for MIT's Altair computer. Others, such as the 1973 French-made Micral microcomputer even featured a cathode-ray tube based "TV Typewriter" with alphanumeric characters visible on a television monitor, although only 500 copies were sold. [13]

Additionally, personal computers were slowly beginning to conquer a new market. Although monitors slowly evolved in terms of cost, size and color gamut, and typewriter-like interfaces were replaced with softer keyboards, it wasn't until Apple's 1982 Lisa and 1984 Macintosh PCs that a new paradigm in user interfaces came about: Instead of a command line interface, users were presented with GUI featuring windows and graphical icons in conjunction with a new pointing device called "the mouse", which controlled an on-screen cursor. Windows and Lisa OS/Mac OS would become competitors in the field of personal computing until today. [13]

In 2007, Apple introduced the iPhone, a mobile computer based entirely around touchscreens. Ignoring performance enhancements, many software upgrades and dimensional changes in the form of tablets and foldable laptops, the iPhone features one of the most modern user interface methods of today. [13]

UXD and Graphical Design

After having defined the hardware, the second part of UI design is software. One of the most popular frontend web design libraries is Twitter's Bootstrap, offering interface designers a large selection of elements for various web interface tasks including navigation menus, layered panels and accentuated notifications. In terms of design rules, guidelines such as Google's Material Design and Microsoft's Fluent Design System serve as guidance for the design of touchscreen-based mobile interfaces with a focus on eliciting joy from users while maintaining consistency. These systems of rules may also be affected by generations of products that alternate between the simplicity of flat design and the intuitiveness of skeuomorphism.

Measuring Improvement

Frequent paradigm shifts in the evolution of user interfaces make it difficult to objectively evaluate whether user interfaces are getting ”better” over time. Jakob Nielsen laments that ”statistics on accidents per person-kilometer are readily available to chart progress in automobile and highway design [...] In contrast, usability has not been measured over the years, and it is not even certain that such measures would have been meaningful, had they been made.” [16]

In [17], however, Kujala et al. describe a novel method of evaluating the quality of a product's user experience over a period of three to twelve months in correlation to customer loyalty. They introduce this measure as a remedy for the high costs and intrusive nature of other long-term evaluation methods such as the Experience Sampling Method or Day Reconstruction Method. The method entails asking subjects to simply draw a line showing their opinion of a product over time in terms of the quality of experience in general, attractiveness, ease of use, utility, and the degree of usage of the product, as well as answering questions and making qualitative statements about features that influenced their experience. [17]

2.1.2 Affect-Based Computing

SitAdapt is one of several approaches to incorporate the use of emotions in computing, however the idea of utilizing a user's emotion isn't new. Rosalind Picard presented several related scenarios, models and considerations in her 1997 paper ”Affective Computing” [18], where she described the perceived evolution of computers to not only perceive, but also bear emotions in order to better assist humans and make decisions.

Explaining that emotions pull ”the levers of our lives”, she describes how human beings rely on emotions to be able to make decisions and carry out actions, stating ”the neurological evidence indicates emotions are not a luxury; they are essential for rational human performance”. In order to demonstrate the possible applications of affective computing, Picard proposes a scenario of a computerized piano teacher trying to hold a pupil's interest and advancing their musical capabilities by finding a balance between challenge and contentness. ”Whether the subject matter involves deliberate emotional expression such as music, or a 'non-emotional' topic such as science, the teaching system still tries to maximize pleasure and interest, while minimizing distress.” [18]

Picard also argues that ”interface agents” (applications that present the user with some kind of graphical UI) shouldn't simply customize the content to match the interests of a user, but rather customize it to match the user's emotional state. Such considerations are not dissimilar to the pursuits of our system.

The Difficulties and Dangers of Affective Computing

In her essay, Picard goes on to explain why some forms of automated emotional recognition aren't always effective, describing how pattern-recognition complexity increases when faced with an increased number of categories, as might be the case when a face is not displaying singular ”pure” emotions. The details of feature-based emotional classification are beyond the scope of this thesis, although possible shortcomings in SitAdapt's input data are obviously of consequence to the rest of the system.

Additionally, Picard warns of the dangers of affective computing, including the ability to manipulate

emotions or having computers become unpredictable in terms of their decision-making. Of course, this would only be the case for computers without the ability to express emotion, as is the case for Rosalind Type I and III computers. This thesis focuses on Rosalind Type III affective computing, wherein computers have the ability to perceive, but not to express affect (see Table 1). [18]

Computer	Cannot express affect	Can express affect
Cannot perceive affect	I.	II.
Can perceive affect	III.	IV.

Table 1: R. Picard's Types of Affective Computers

In [19], D. Fehrenbacher describes how emotions detected via facial analysis can influence decision-making and knowledge-sharing processes, increasing the effective use of an organization's data design. Fehrenbacher references Forgas' Affect Infusion Model, which describes how social judgment (that is, the perception and evaluation of ideas in an environment) is derived either from a higher or lower affect-based factor, according to the type of strategy involved. [20]

While not necessarily affect-based, a similar approach is taken in [21], wherein data gleaned by manipulating task demand and observing a subject's pupils is used to introduce a quantitative measurement of task demand into information systems. This kind of data can be used to influence "adaptive decision support systems" in a more specialized manner than SitAdapt 2.0 currently allows. However, it should be noted that these approaches to influencing a backend system's decision-making and business logic choices are different from SitAdapt's focus, which consists of providing a framework for adaptive user interfaces.

2.1.3 Complex Event Processing

Complex event processing (CEP) has been described as "computing that performs operations on complex events, including reading, creating, transforming, abstracting, or discarding them. Note: CEP ultimately creates complex events even if some or all of the source events are simple events" [22]. Complex events are described as being a collection, representation or description of one or more other events. Examples include the 1929 stock market crash as a complex event that encompasses several thousand trading events, a "CPU instruction" representing several actions on a register level, or an e-commerce application's "checkout" event being the amalgamation of several shopping cart and business logic related events. Note that some observers or applications regard certain complex events as simple events. [22]

[23] describes five application categories of complex event processing with examples:

1. "Real-Time Enterprise" processing handles business logic. For example, as part of a stock market trading application, the handling of a trade action should cease immediately if the request has been withdrawn.

2. "Active Diagnostics" processing identifies the sources of errors by representing symptoms as simple or complex events.

3. ”Information dissemination” processing handles subscription-based notification systems. A combination of events, such as when a certain stock has changed in value over a certain timeframe, triggers a notification.

4. ”Business Activity Monitoring” processing evaluates certain business logic indicators for abnormal activity. It detects, for instance, when a product shipment deadline was not met.

5. ”Prediction” processing involves the prediction of problems in the future and trying to proactively counter them.

SitAdapt 2.0 allows users to specify a set of conditions on which to carry out a set of actions. The triggering of one of these conditions can be considered a simple event, and the boolean operator-designated combination (union or intersection) of these conditions can be considered a complex event. Our system’s handling of incoming data events can therefore be considered ”Real-Time Enterprise” complex event processing.

Perhaps the topic can be more easily understood by analyzing the main problem it’s trying to solve: Chaos. [24] calls attention to the expansion of activity in the areas of global electronic trade, the emergence of various dynamic information-gathering systems and the increasing number of cyber warfare threats, all of which are requiring an increased amount of resources in terms of situational analysis. An unavoidably increasing amount of heterogenity in data, scattered across different ecosystems must meet rising expectations in terms of speed and reliability. These challenges can only be overcome by intelligent event processing facilities.

2.1.4 Software Patterns

In order for the system’s Adaptation Component (see section 3.2.2) to be able to make decisions about what parts of the interface to change and how, it requires instructions about how to do so.

The modern idea of using special patterns to define specific problem-solving approaches for specific problems was formally introduced in [25] as the centerpiece of a proposed manual of architectural design theory. In it, Alexander et. al introduce an organized way of recording and presenting established architectural means and proposals, in conjunction with its sister book, ”The Timeless Way of Building” [26].

The authors describe a language containing an alphabet of patterns: ”Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. For convenience and clarity, each pattern has the same format. First, there is a picture, which shows an archetypal [sic] example of that pattern. Second, after the picture, each pattern has an introductory paragraph, which sets the context for the pattern, by explaining how it helps to complete certain larger patterns.”

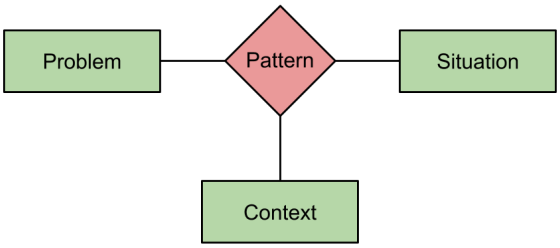

Figure 3: A pattern is a relation consisting of a problem, solution, and context

As presented in [27], ”there is no mention of specific details [...] These implementation details are left to the designer, allowing different instances of the same pattern solution.” This usage of patterns gained popularity in a 1987 conference on object oriented programming (OOP), culminating in the modern-day ”bible” of software architecture, ”Design Patterns: Elements of Reusable Object-Oriented Software”, also known as the ”Gang of Four” book for its authors Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides [28].

Situational Patterns

Our system depends on a set of administrator-defined rules that define what adaptations to make to the frontend system and the circumstances under which they are to be triggered. Although these definitions can be called behavioral rules [29], the question of whether or not they can be considered patterns is debatable, as patterns such as those exemplified in [25] can be *reused*, whereas our rule set offers no kind of flexibility.

UI Patterns

Although our behavioral rules might not involve patterns, some of our adaptations do, in the form of UI patterns. Like Alexander’s architectural patterns, UI patterns are solutions to problems within a certain context. One such repository is [30], listing dozens of graphical user interface solutions in various categories, including collecting user input, implementing navigation, presenting data, encouraging behavior and garnering user engagement.

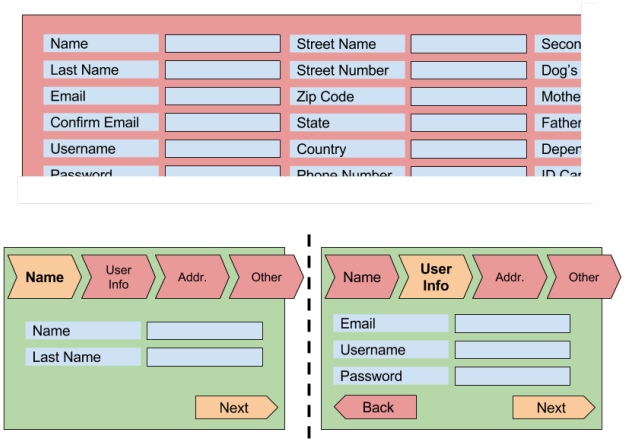

Figure 4: A wizard is a UI pattern that splits one large task (top) into multiple smaller steps (bottom)

One user input scenario occurs when a user needs to enter a lot of data in order to complete one task, although the task could be broken down into multiple steps which build on each other. The theoretical solution for this ”problem” is a wizard, a splitting of the form across multiple pages. Future versions of the SitAdapt 2.0 system might a prompt frontend application to display a wizard instead of a complex interface, depending on the user's needs.

Software Patterns

The most prevalent *software* patterns of SitAdapt 2.0 are the observer and the publish-subscribe patterns. In the former, a subject is added to an observer's list of clients to be notified when an event happens. This is the case for RethinkDB changefeeds, wherein a registered callback is triggered when the observer has new data (as a consequence of a conditional query's results changing). The publish-subscribe pattern, on the other hand, includes a message broker that facilitates transmitting new data from the publisher to subscribers, meaning the sender and receiver of the message don't know about each other [31]. This is the case when SitAdapt 2.0 sends a message to the Websocket server containing an adaptation command for the frontend. The subscribed frontend's websocket client receives the message and processes the command accordingly with no knowledge of the sender.

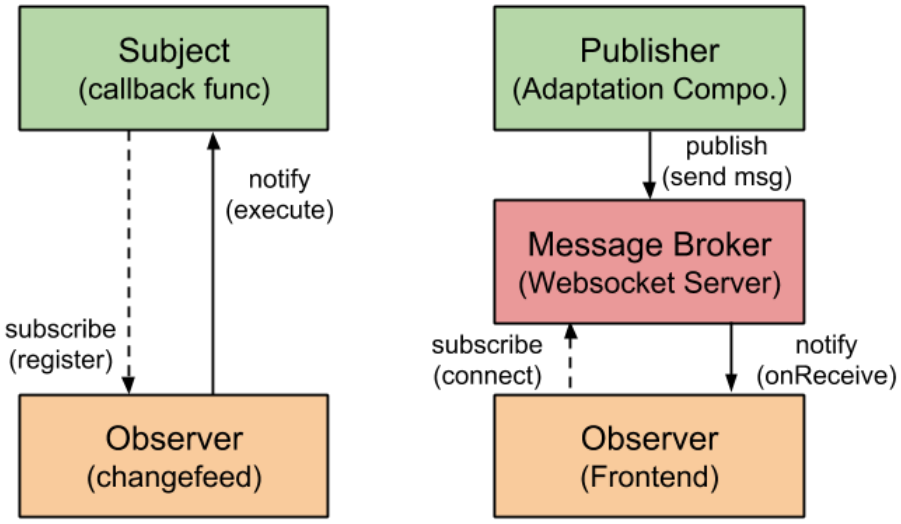

Figure 5: Observer Pattern (left) and Publish-Subscribe Pattern (right)

2.2 Application

This section will give an overview of the sensors that we chose to integrate in the SitAdapt 2.0 system and our reasoning for doing so. Additionally, some of the employed web and server technologies will be introduced, in addition to our database selection.

2.2.1 Sensor Technologies

SitAdapt 2.0 receives data from four input sources, which we call sensors. They are an infrared-projection based eye tracker from Tobii, the webcam-based emotion-classification system called Noldus FaceReader, the biosignal-focused multi-sensor device Empatica Wristband E4, and a modern web browser. Detailed information about the data we collect from each device can be found in the appendix.

Eye Tracking

The Tobii Eye Tracker provides information about where the user is looking, and their pupil dilation. This information can be used to measure how strenuous an activity is for the user in knowledge-sharing systems [21], analyze the effectiveness of a user interface in terms of usability, and evaluate the user's current optical focus and level of attention.

Figure 6: The Tobii X2-60 Compact Eye Tracker

While early usability research efforts employed cameras that recorded a user's eyes and required researchers to manually correlate eye movement with the present interface state, eye tracking technology has advanced steadily since then. Tobii's screen-mounted X2-60 records its data by projecting infrared patterns onto the user's cornea, and recording the reflections via image sensors (together with other features, such as pupil dilation). Proprietary algorithms can then use this data to infer each eyeball's position in 3D space, which is then used to discern the user's current gaze point, i.e. where he/she is looking on the screen. [32]

Eye tracking can aid application developers both as a retrospective tool (seeing what elements of a specified design a user focuses on, or in what order visual cues are processed) and as a real-time input device (what's the user looking at and is he focused?). Our primary focus is on live data that can be used to have the application react in real-time.

Interpretation and Accuracy of Data

However, as detailed in [33], turning eye tracking data into useful knowledge that applications and

interface designers can act upon is a challenge: ”Eye tracking equipment is still far less stable and accurate than most manual input devices.” Also, ”like other passive, lightweight, noncommand inputs (e.g., gesture, conversational speech), eye movements are often non-intentional or not conscious, so they must be interpreted carefully to avoid annoying the user with unwanted responses to his actions” [33]. Other problems with eye tracking research include the ambiguousness of certain terminology between papers like ”fixation”, although usually referring to ”a relatively stable eye-in-head position within some threshold of dispersion over some minimum duration” [33].

SitAdapt 1.0 divided its prototypical frontend application’s user interface into large rectangles (called panels) that made it easier to find out which part of a form was being looked at at any given time. Due to the relatively wide scope of SitAdapt 2.0 and the separation of frontend and backend development, there has been limited testing regarding the accuracy of the Tobii Eye Tracker and to what extent a similar panel-based approach would be necessary for our application’s purposes. Current implementation details are available in Section 3.2.4.

Face Reading

Noldus FaceReader is an application that classifies facial expressions. Using a webcam, a video or even a still image as an input source, the program extrapolates information regarding the presence of emotion such as anger, happiness and surprise using a multi step process. It also provides data about the user’s gaze direction, their facial features, and estimates about their age and ethnicity.

Figure 7: A webcam used as an input device for Noldus FaceReader

For every incoming frame, Noldus FaceReader first tries to detect the face via the Viola-Jones algorithm (see [34]). Next, it creates a three-dimensional representation of the face using several hundred points. Finally, if face detection was successful, it uses a an artificial neural network trained with an advertised 10,000 points to classify the emotion. If face detection fails, FaceReader resorts to classifying emotions on a pixel-based process called Deep Face classification.

An important part of using Noldus FaceReader in a production environment is making use of its calibration feature, as ”for some people, FaceReader can have a bias towards certain expressions.” After calibration, ”the facial expressions are more balanced and personal biases towards a certain expression are removed” [35].

We can use the information provided by Noldus FaceReader to detect when a user is annoyed, neutral, or visibly satisfied within their current context in the application (the context is the URL of the

site currently being displayed). Depending on how the adaptation rules are defined, the system can execute actions to mitigate or exploit the user's emotion. Application administrators can also make use of data containing the subject's physical characteristics, such as the state of the mouth (open, closed), eyebrows (raised, lowered, neutral), beard (full, none), age range and demographic.

Pulse Measurement

The Empatica E4 Wristband is a mobile sensor platform which is attached to a user's wrist. It provides information about arm motion, cardiovascular activity, skin temperature, and the skin's electrical conductance. The wristband offers two modes of recording this information, either streamed wirelessly in real-time, or written to the embedded device storage for offline use. [36]

Figure 8: The inside of the Empatica E4 Wristband

Two sets of applications correspond to the two operating methods of the device. If not connected to a streaming client, information is written to the wristband's internal memory for offline viewing. After connecting it to a computer with the Empatica Manager app running, data is transferred to Empatica's cloud-based storage and viewing platform, Empatica Connect.

For streaming use, the device uses the Bluetooth low energy (BLE) protocol to send data either to a mobile application running the Empatica Realtime app, or to the Windows-based desktop application Empatica BLE Server. We make use of the latter system, using a modified version of the C# console application-based Empatica BLE Client. [37]

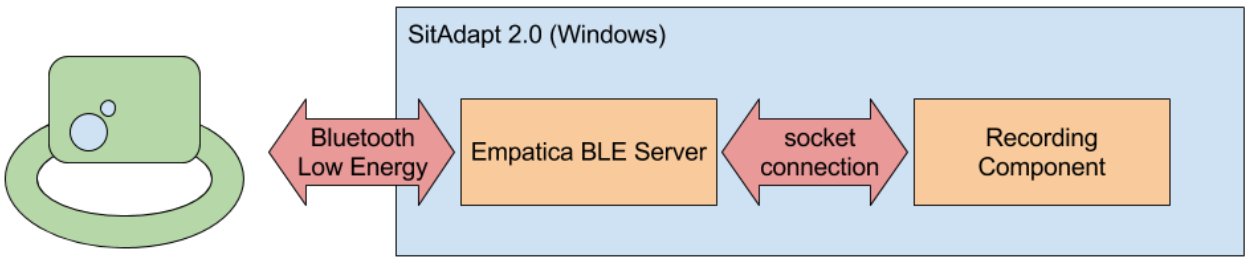

Figure 9: Overview of the Empatica E4 Wristband connection

Our reasoning for including the Empatica E4 Wristband in our system was to add an additional data source for establishing the attentiveness and frustration level of the user. We chose the E4

Wristband for its streaming capabilities, albeit some of the data transmission could be more refined (see ”Empatica Wristband” under Section 3.2.4 regarding the steps necessary to clean up incoming device data).

Electrodermal activity

One of the E4 Wristband’s sensors measures the electrical resistance of the skin multiple times per second. Also called electrodermal activity (EDA), this value is similar to the terms skin conductance, galvanic skin response (GSR), electrodermal response (EDR), psychogalvanic reflex (PGR), skin conductance response (SCR), and skin conductance level (SCL) [38].

[39] states that electrodermal activity ”is arguably the most useful index of changes in sympathetic arousal that are tractable to emotional and cognitive states as it is the only autonomic psychophysiological variable that is not contaminated by parasympathetic activity”, referring to the involuntary nervous system’s tendency to influence heart rate, intestinal and glandular activity and sphincter muscles [40]. EDA also serves as an indicator of attention, as well as subconscious awareness of threats, anticipation or delight.

One problem involved with processing EDA data is the unreliability of its absolute value, or even a user’s recorded baseline value at the beginning of a SitAdapt session. As explained in [39], this is due to the EDA’s ”constantly moving baseline”, meaning useful information can only be gleaned from relative changes to the data.

2.2.2 Cross-Component Communication

One of the most critical input devices of the SitAdapt 2.0 system is the user’s web browser, as it provides information about the user’s current context, identified by the current URL of the application. Additionally, information about user input is provided in the form of mouseMove and mouseClick events, which include data about the position of the cursor and a list of HTML elements items that a user clicked on.

To facilitate the communication between disjoint components of the system (like the browser, recording component, database writer, rule editor, live observer), a flexible real-time protocol was required. Advancements in browser and server technologies have brought about a variety of network traffic solutions for the transmission of time-sensitive data, which will be briefly described here.

HTTP

HTTP, the Hypertext Transfer Protocol, dating back to 1990, describes a stateless communication protocol for ”hypermedia information systems” [41]. Requesting a resource via HTTP happens as follows:

1. A client application sends a request to the server for a specific item.
2. The server processes the request.
3. The server sends back a response.

Starting with HTTP/1.1, persistent connections were made part of the standard, which allow clients to keep TCP connections to the server open even after having received a request. This allowed

for the pipelining of requests (that is, sending multiple requests to the server without waiting for a response) via a single connection, saving network and CPU resources. [42]

AJAX Polling

In AJAX Polling, a client repeatedly sends requests to a server to see if new data is available. In practice, the following happens:

1. A web browser requests and receives an HTML page from the server.
2. A script on the page polls the server at regular intervals (using an XMLHttpRequest, for instance).
3. The server sends back a response.

Ajax-Long Polling

Instead of polling the server in regular intervals, long polling involves keeping a network connection unresolved until data arrives. [42]

1. The browser requests an HTML site via HTTP.
2. A script on the page polls the server once.
3. The server neither responds to the request nor terminates the connection. Only when new data arrives does the server complete the request with the relevant data.
4. Having received data, the script once again sends a request to the server

Server Sent Events (SSE)

Server Sent Events require event-based logic to be running on the server. They represent an improvement over long polling on account of not having to close connections. Cross-domain requests aren't as simple to implement, however. [42]

1. The browser requests an HTML page normally.
2. A script on the page opens a connection to the server and registers an event listener for incoming data (often expecting a specific message header).
3. When the server has new data available, it sends it back in the form of an event and keeps the communication channel open.
4. The client's event listener handles incoming data and stays ready to receive more messages.

Websockets

Websockets are similar to Server Sent Events in terms of the connection state, however a client can not only receive events/messages, but also send them. Additionally, they function well with external domains. [42]

Unfortunately, certain issues arose when trying to add websocket functionality to our system's Recording Component. As explained in [43], the .NET runtime environment provides no support for

the implementation of websockets. As solution, we chose to use the third party library WebSocket-Sharp.

Alternatives to using websockets include the free multi-platform solution Socket.IO, which provides fallback support to older web browsers and applications via long-polling instead. Despite its features (including the ability to define individual event listeners by the message header), a seeming decline in community support discouraged us from using it.

2.2.3 Frontend Development and Framework Selection

Although the focus of SitAdapt 2.0 is on adapting an existing web application, we elected to provide program administrators with an easy-to-use configuration interface in the form of a different web application. The primary goals of this configuration tool were the ability to define the adaptation conditions and actions, and the ability to start a session (starting a session entails subscribing to changes in the database and starting to carry out adaptations). Future versions of the system may show diagnostic information, incoming data, detected events and the execution of adaptations.

Dynamic Rendering Requirements

The configuration of the system's rules can be very complex, with certain parameters only being applicable when certain options are selected. For instance, it would only make sense for the system to require a textual argument for an action if that action presents a prompt or message to the user. For conditions with qualitative values, such as ethnicity, comparative operators such as ”greater than” would not be appropriate. In other cases, the available options for a certain comparison should be limited, such as for the condition ”Right Eye”, which can only bear the value ”Open” or ”Closed” in the current system. In order to make the interface as intuitive as possible, we sought a way to conditionally render parts of the interface. Our framework candidates were AngularJS, VueJS, and ReactJS, the latter being our framework of choice due to its low overhead and large amount of community support.

Our first efforts into designing the interface involved using plain HTML and JavaScript, and required a lot of conditional logic for hiding and displaying editor elements. When the editor's list of requirements grew more complex, our frontend eventually turned into a not-easily-maintainable monolith of code that created difficulties in handling even a small list of items. For example, in addition to a comparison with static numbers, the comparison of values to dynamic runtime-created variables such as a user's baseline emotion was deemed a desirable addition. Thus, the early design was dropped in favor of a reimplementation with React.

Capabilities of ReactJS

ReactJS (see [44]) can be used for many of the same purposes as AngularJS but comes with less overhead (there are no directives, services, factories, and other included modules that might not be utilized by the application). Its most attractive features include the declarative nature of its views (with two-way data binding between application state and form), its component-based architecture that lets developers create small, reusable elements using a new Javascript- and HTML intermixed language, and its lack of technology stack assumptions. Another attractive feature was its easily installable development server in the form of the cloneable repository ”create-react-app”.

Two-way data binding describes an implementation property wherein changing the view will change the state and vice-versa. The decades old MVC (model, view, controller) concept usually describes a software pattern featuring a view that generated to show information from a model, with a controller used as a proxy to facilitate changes to the model via the view. [28] React can instead be argued to present a MVVM model (model, view, view-model) wherein application state (the model) is part of the document object model, and the view is bound to the data directly via event handlers (the view-model). No facilitating or processing of data to external controllers takes place, and only the view is recalculated.

2.2.4 Server Framework Selection

In theory, SitAdapt 1.0 was already using a working server framework, namely ASP.NET, Microsoft's app and web service framework. In practice, it was only taking advantage of the service's script packaging and bundling capabilities, and its ability to serve files.

For SitAdapt 2.0, we elected to go a different route, namely using the NodeJS module-focused, JavaScript-based server framework. NodeJS lends developers the unique ability of programming the backend and the frontend of a web application using the same language, making it especially comfortable to use for data structures that are native to the web, especially JSON (JavaScript Object Notation).

NodeJS's modular structure can often contain a lot of dependencies (React's create-react-app dependencies folder, for example, has more than 23000 items using more than 120MB), but this modular structure nevertheless attempts to follow the respected UNIX philosophy of ”doing one thing and doing it well” [45]. NodeJS exists hand in hand with its most popular package manager, npm, featuring various abilities as standard, including automatically fetching dependencies, tracking of dependency versions, and testing.

SitAdapt 2.0 makes use of the npm packages ”Express” (to serve as hosting and routing middleware), ”http” (for simple GET requests), ”url” (for parsing the location of network-attached resources), ”WebSocket” (to provide the WebSocket server), ”rethinkdb” (to setup a connection to the database and make use of Rethink's self-implemented Javascript-based query language) and others.

2.2.5 Database Selection

Several database systems were considered for the design of the new SitAdapt system. One of the desired properties of the database was the ability to deal with JSON structures without the need for special parsing (in order to be more compatible with the Javascript-based server framework), making database management systems like MongoDB, CouchDB and MySQL 5.7+ viable candidates.

In the end, RethinkDB was chosen to be our database due its unique event-driven architecture. Called ”Changefeeds”, RethinkDB offers the ability to define a query to the database that, when the results for that query update, triggers a callback function set by the user. The following code snippet exemplifies a query written in Rethink's syntax that gives console output if, during the last two seconds, there is an increase in the number of database rows that contain an ”angry” value greater than ”0.3”.

```
r.table("FaceReader")
.filter(r.row('time').during( r.now()-2.000, r.now() ))
.filter(r.row("angry").gt(0.3))
.count().gt(0)
.changes().run(conn, function(err, cursor) {
  cursor.each(console.log("User is angry; change background color."));
})
```

These changefeeds are the centerpiece of SitAdapt 2.0's Situation Analytics component, in that a function is triggered as a result of some defined conditions changing. Of course, the code snippet only shows a static, hard-coded situation whereas the final application bears a dynamically-created changefeed.

RethinkDB also features the ability to use multiple indices to optimize storage of data, nested fields to make full use of flexible JSON schemas, map-reduce capabilities for powerful list- and set-based operations, asynchronous connections, many data types including dates, binary data and geospatial info, sharding, scaling, replication, permissions, version migration, multiple third-party administration tools (for example, we used Chateau - https://github.com/neumino/chateau - when RethinkDB's own web-based application fails to load), and many other deployment, integration, and architecture-related features. [46]

2.3 Related Work

2.3.1 The SitAdapt Architecture

SitAdapt 2.0 is related to, but explicitly NOT a reference implementation of, the official SitAdapt architecture as presented in [47], therein described as a runtime-support architecture making use of PaMGIS (see section 2.3.3). The architecture's models and HCI patterns can be derived from both the evolutionary process of the system's development, and from repositories that bear the resulting patterns of previous development phases. Adaptation decisions are then based on the domain model of the target application together with the context and emotional state of the user.

As a motivation, Herdin et al. lists the genesis of new user analysis capabilities in terms of emotional state, mood and stress level analysis, allowing for the possibility of changing some aspect of a software in real-time as a response to affect-based properties, such as in gaming and e-commerce applications [47].

2.3.2 Model-Based UI Development

Modern user interfaces have to deal with a lot of different factors in order to achieve their goal of having users complete tasks in an efficient manner. In [48], the World Wide Web Consortium (W3C) lists these factors as the following:

1. Different kinds of users in terms of preferences, experience and cultural background
2. Different kinds of hardware platforms and interaction systems, frameworks and tools
3. Different kinds of environments (noisy, on the move, etc)
4. Different contexts of use

Additionally, [49] claims that adapting UIs at runtime to unforeseen factors poses a significant problem, as some elements of classic design theory are thereby no longer applicable to the design process. One cited example is ”empirical measurement”, referring to data ascertained during user tests of a static design, which can inform developers about usability problems such as misleading descriptions and poorly sized navigation elements, as well as measurements regarding user satisfaction.

One solution to mitigate these development concerns in dynamically changing interfaces was the creation of the model-based CAMELEON Reference Framework (CRF) [50], which sought to formalize the combination of context of use (device-specific implementation), usability and application-specific models.

Abstraction Levels

An important part of CRF-related model-based user interface development (MBUID) is defining models for interaction logic at various abstraction levels. At the highest level of abstraction, ConcurTaskTree (CTT) notation is used to document both user and system tasks. Next, an Abstract User Interface (AUI) denotes interaction possibilities without specifying the platform or method of operation. The Abstract UI model is materialized by the Concrete UI model (CUI), which does include platform and modality information, resulting in a mock-up of the graphical user interface. In the last step, a Final User Interface (FUI) is established, which may take the form of a finished website with inputs that execute system functions. [49]

As this process is monodirectional, the final implementation has no way to refer to the results of the previous steps. [49] goes on to explain how runtime architectures (in our case, SitAdapt) can extend the original MBUID process by keeping the CTT, AUI and CUI models in the final product, allowing the Final User Interface to react to changes in interaction models and the context of use. Another aspect of this extension is the inclusion of metadata in each of the FUI’s elements, in order to understand the changes that led to the current state of the interface.

2.3.3 Pattern-Based Modeling and Generation of Interactive Systems

Motivated by the potential to reduce time-to-market and development costs, PaMGIS (Pattern-Based Modeling and Generation of Interactive Systems) hopes to partially automate the construction of UI models that can be dynamically modified to fit various contexts of use.

PaMGIS is based on the Cameleon-Reference Framework (see [50]), itself an architecture for developing UIs based on models and patterns. While the CRF architecture already describes a process by which an abstract interface is turned into a concrete interface by a series of irreversible reification steps with immutable inputs, PaMGIS describes a system wherein runtime decisions can also influence a concrete UI. [47]

The CRF suggests that all MBUID models should include the following components:

1. domain model (user tasks and UI-relevant data elements)
2. context-of use (user, platform, environment model)
3. adaptation (evolution and transition model)

HCI Patterns describe implementation solutions for UI designs, and, just like in section 2.1.4, they represent a relation between a problem, a solution, and a given context. Multiple HCI patterns form a pattern catalog, and a pattern catalog for a specific domain can be considered to be a pattern language. A proposed formal style description is the XML-based Pattern Language Markup Language (PLML) consisting of various attributes such as the aforementioned triad of domain, context and solution, as well as diagrams, literature, implementation and related patterns. This is superceded by the PaMGIS Pattern Specification Language.

PaMGIS seeks to combine both MBUID models and HCI patterns to form graphical user interfaces, as well as incorporating a usability test-based feedback loop to influence the original patterns and models. The framework largely follows the steps outlined in section 2.3.2 with the aforementioned integration of patterns. Although a method for carrying out (and measuring the effectiveness of) UI adaptations is not present in the framework, it does offer the ability to record the evaluation of each UI model pattern. [51]

2.3.4 SitAdapt 1.0

Called SitAdapt 1.0 in this thesis, the first version of the SitAdapt system (see [5]) exhibited a different architecture compared to SitAdapt 2.0. While a separation of concerns between the C#-based recording component and the reactive frontend web application was apparent, the similarities stop there. All of the logic for deciding the properties and location of adaptations to the frontend was embedded in the so-called ”directives”, ”services” and ”controllers” of the AngularJS web application, small Javascript-based modules executed by the browser that handle the presentation logic of the site. In order to initiate a change with the old system, the following actions would need to occur:

1. The C#-based backend receives new data from input devices.
2. A service in the AngularJS web application requests new data from the local server (this occurs every 1000 milliseconds).
3. The ASP.NET-based self-hosted API of the .NET application returns the newest FaceReader and EyeTracker data in the form of a JSON Object (Note that this implementation detail also requires Administrator Rights to run the application).
4. A directive in the web application processes the data, changes the local state to record what panel is being looked at, and the user’s emotion while doing so.
5. A different directive checks if certain hardcoded conditions are met (e.g. the user is focused on a panel and is displaying either a happy or an angry expression for the fourth time).
6. If this is the case, a service function is called to change the class of certain elements on the page. This causes the browser to modify the design attributes of the page to those in a static CSS file.

The following code listing illustrates how SitAdapt 1.0’s adaptation analysis and application was executed (with certain parts omitted or spliced from external files). Note that when the user is angry (as a [fabricated] result of hiding the form to input a destination), the destinationbox variable is updated to retroactively render the input box:

```
if (dominantExpression(routeAreaEmotions) == "Angry" {
  emotionState = "Angry";
  scope.destinationbox = true;
}
else if (dominantExpression(routeAreaEmotions) == "Happy") {
  emotionState = Constant.EMOTION[2];
}
/* .... /*
var design = uxDesignService.change(emotionState);

//in uxDesignService:
this.change = function (emotionState) {
  $(".panel-primary").addClass('headingColor' + emotionState);
  $(".panel-primary").addClass('panelColor' + emotionState);
  $(".panel-primary").addClass('panelBody' + emotionState)
  $("h1").addClass('viewheading' + emotionState);
  $(".btn-primary").addClass('buttonColor' + emotionState);
  $("body").addClass('image' + emotionState);
}
```

SitAdapt 2.0 has a slightly different method of operation:

0. Let the system administrator define conditions and actions.
1. The C#-based backend receives new data from input devices.
2. A controller in the AngularJS web application connects to the Websocket server and waits for new messages.
3. The Analytics Component (a subcomponent of the Database Writer) subscribes to change-feeds in the database using the user-defined conditions.
4. When a condition is evaluated to be true, the Situation Analytics Component sends a notification to the EvaluationAndAdaptation Component about the detection.
5. The EvaluationAndAdaptation Components determines what, if any, actions to carry out for the given condition's rule, and sends a message to the web application via the Database Writer containing an adaptation command.
6. A controller in the web application processes the message and applies the transmitted styling changes, adds and removes elements from the site, or changes the application state.

Other limitations of SitAdapt 1.0 include its reliance on an old version of the Tobii Software Development Kit, namely the Tobii Analytics SDK 3.0. While it featured a .NET application integration of the connected EyeTrackers (a list of EyeTrackers was updated automatically in a window when devices were added or removed), it did not include detailed information about the validity of each piece of data. A limitation that both the old and the new version of SitAdapt share is the requirement of having to manually set up a recording in Noldus FaceReader prior to starting a session.

It's worth mentioning that SitAdapt 1.0 also included a desktop application for the administration of the system. Many of the actions that a SitAdapt administrator would have needed to carry out were automated in SitAdapt 2.0, including the connection setup with Noldus FaceReader and the Tobii EyeTracker. SitAdapt 1.0 also included a small manually-activated calibration program for the eye

tracker, a task that was delegated to the proprietary Tobii Eye Tracker Manager software in SitAdapt 2.0.

Whereas SitAdapt 1.0 was limited in function by its hard-coded, highly-coupled situational adaptations that only allowed changes to pre-designed elements, the aim of the new system was to facilitate low coupling by abstracting the situation analysis components from the user interface.

2.3.5 Other UI Adaptation Systems

"Toward a UI Adaptation Approach Driven by User Emotions"
In [52], Galindo et al. note that emotional factors tend to only be considered at design time, and propose their own runtime UI adaptation system as a way to dynamically adapt to context-of-use changes, specifically the user, in terms of the emotions "positive", "negative" and "neutral". Their approach is similar to SitAdapt 2.0 and involves sending UI parameters to the frontend system, albeit in a ten step process:

1. A so-called "Inferring Engine" monitors sensors for new data.
2. When new data arrives, a new "context of use" (a set of user, platform, environment) is inferred.
3. An "Emotion Wrapper" collects [additional] sensor input and sends it to an external emotion classification tool for processing.
4. The tool (for example, Noldus FaceReader, or Affectiva's AffDex) sends analysis data back to the Wrapper.
5. The Wrapper determines the valence of the emotion (negative, positive, neutral) and sends context of use information to the "Adaptation Engine".
6. The Adaptation Engine delegates the task of determining the ideal emotion-based UI adaptation to an adaptation rule component and delegates the task of finding the ideal UI variant for the screen size to a variant-description-component.
7. The adaptation rules component determines the best UI parameters for the given emotion (e.g. background-color = light-yellow for a positive emotion).
8. The UI variant description component selects the ideal UI variant for the given dimensions.
9. The "Interactive System" requests a new user interface.
10. The size variant and UI parameters are sent to the frontend system, which renders the final website to the user.

For a single two-minute trial session of the system, Galindo et al. demonstrated that the cumulative average of negative emotions decreased for each iteration of the UI within that session, inferring that the changes achieved their intended goal. However, the authors do note that "humans seem to be inconsistent in their rational and emotional thinking evidenced by frequent cognitive dissonance and misleading emotions", inferring that the current implementation may still be partially ineffective.

"Adaptive Intelligent User Interfaces with Emotion Recognition"
In [53], Nazos presents an event-based adaptive virtual driving interface. Utilizing data gained from a BodyMesia SenseWear armband and a camera, in conjunction with movie clip experiments and a virtual reality environment, Nazos first derived physiological patterns of emotions and then implemented a Bayesian Belief Network to respond to users' negative emotions in a virtual driving environment.

Nazos's adaptive interface uses an event-based model to identify the optimal adaptation for a given user, based on the recognized emotions of the user (anger, panic, sleepiness, frustration), established personality traits, age, gender and frequency of each experienced emotion.

Possible adaptations for the virtual driving interface included having the system suggest a relaxation technique (e.g. in the case of anger combined with a "conscientious" personality trait), making a joke (e.g. for neurotic males below 25), suggesting to stop the car and take a rest (in case of fatigue), changing the radio station, rolling down the window, splashing some water on the user's face, or reminding the user about how their dangerous driving might affect their loved ones. [53]

2.3.6 The OpenSSI Framework

Human-to-human communication consists of more than just spoken words directed at some conversation partner. Rather, it includes the inflections of one's voice, acted gestures, mimicry and other methods. By providing a platform for detecting and interpreting these social signals, the University of Augsburg's "Open-Source-Social-Signal Interpreter" hopes to "[pave] the way towards a more intuitive and natural human-computer interaction." [54]

Wagner et al. cite numerous challenges in creating their SSI platform, including the synchronization and coherent treatment of signals (such as when the sample rates of heterogenous pieces of data lie far apart), the high amount of variability, uncertainty and ambiguity in human communication, the fusing of multimodal data (either at the raw data level, at the level of extracted features, or at a more abstract decision level), and the challenge of real-time processing to achieve a fluent interaction.

The SSI platform uses two data types to handle its processing: streams and events. Data read from sensors enters the platform via streams, which can pass through filters, feature extractors and other manipulation components. Other components detect certain types of activity and can trigger events. In terms of an interface, openSSI by itself relies on XML files as input, although it might be possible to construct a frontend application. A C++ API also provides developers with the ability to create new components. [54] Future versions of the SitAdapt system may interface with the OpenSSI framework to gain additional information about the user.

3 Implementation

This section describes the implementation details of SitAdapt 2.0. Section 3.1 provides an architectural overview of the system, section 3.2 explains each module in detail, and section 3.4 describes a typical walkthrough of the system in terms of a user's inputs and the resulting actions by the system.

3.1 Architectural Overview

The system is comprised of several components that interact with one another in order to record and consolidate sensor data, define and detect situations, and trigger adaptations in a frontend application. Figure 10 visualizes the system's components. This section endeavours to explain the encompassing architecture in an abstract way. Details regarding the collected data, as well as instructions for setting up the system are available in the appendix.

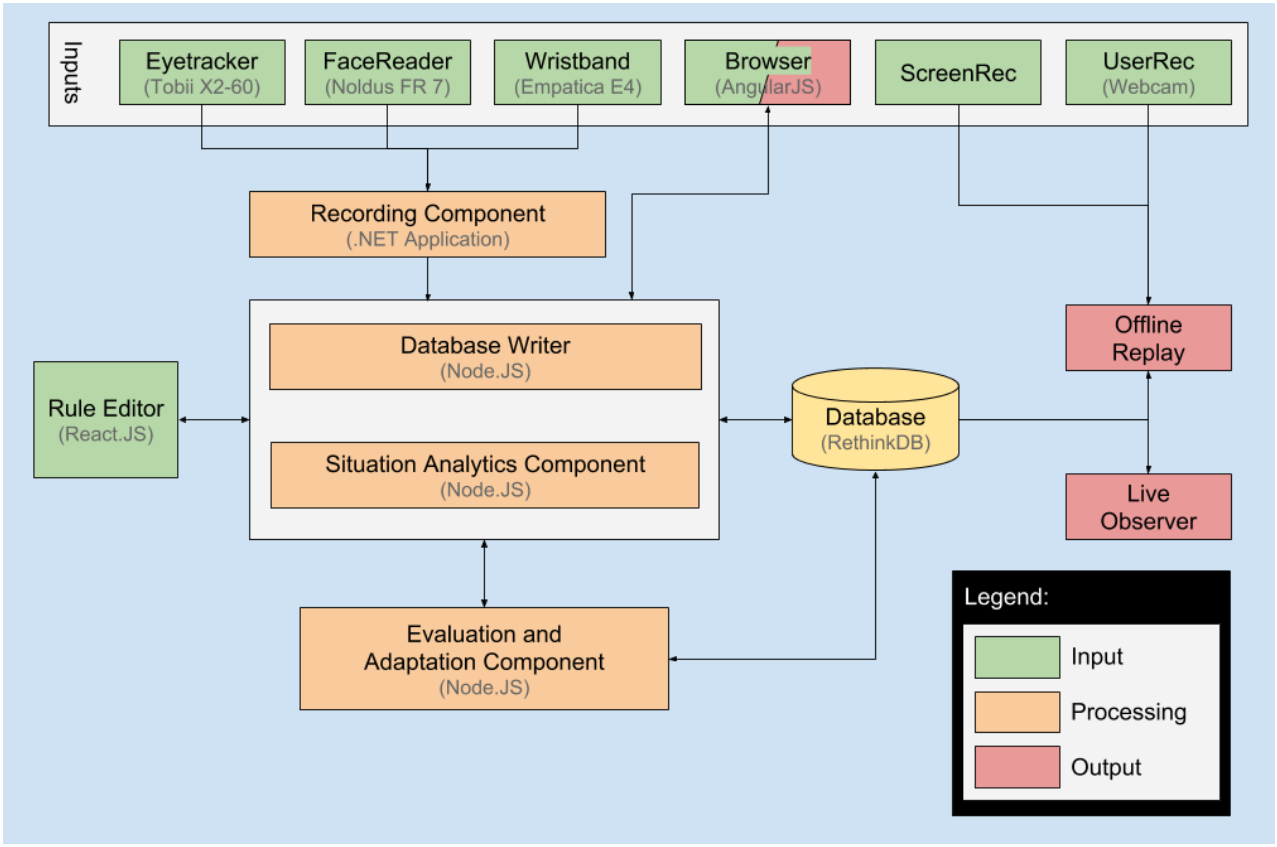

Figure 10: SitAdapt 2.0 System Overview

Handling Input

Three hardware devices and one piece of software form the four "sensors" that SitAdapt 2.0 uses to detect situations. Additionally, a small web-based editor was constructed to allow users to define situations and UI optimizations in the system.

The Recording Component (written in C#) is a .NET-based console application that connects to the hardware components and subscribes to their data streams. When data arrives, it is extracted, validated, timestamped, and repackaged as a JSON Object, whereupon it is sent to the system's NodeJS-based Database Writer, which writes the data to a RethinkDB instance running on the local

machine. The Database Writer acts as a mediator between the Recording Component, the database and the browser-based Rule Editor, with cross-component communication taking place via a WebSocket server hosted by the Database Writer.

Situational Awareness

In this thesis, a set of conditions under a common description is referred to as a situation, and a situation paired with a set of actions is called a rule. The Rule Editor, a small web application, is used to define what actions are to be triggered under which conditions. The Rule Editor is also used to initiate the condition-monitoring process via a ”Start Session” button. When some rules have been defined and the start button is pressed, the ”Situation Analytics Component” (part of the NodeJS script that hosts the Database Writer) subscribes to changes in the database matching the conditions contained by the rules.

User Interface Optimization

When the Situation Analytics Component is notified of a situation detection, it alerts the Evaluation and Adaptation Component. That component looks up the actions that are to be executed for that rule and instructs the web application to carry them out (with the DatabaseWriter again acting as a middleware).

Observation

Two intended components of the system were an Offline and a Live Observation tool that would allow system administrators to monitor the detection of and response to situations from an external computer. While not fully realized and therefore excluded due to time constraints, we present a suggested approach to implementing an observation tool.

3.2 Component Descriptions

This section describes the inner workings and interfaces of each component of the system in greater detail. The components are presented in the order in which they are to be executed by an administrator of the system.

3.2.1 The Database Writer

Assuming an instance of RethinkDB is running, the first and most important component of the system to be executed is the DatabaseWriter, which bears multiple responsibilities. The first of these entails passing data and commands between components (such as sending the contents of the Rules table to the Rule Editor, or broadcasting execution requests from the EvaluationAndAdaptation component to all connected clients). This is accomplished by hosting a local WebSocket server, and by listening to certain messages, as exemplified by Table 2, which describes the signature, description, and approach of various messages that may be received.

Message Type	Description	Handling
(non-JSON)	Any message that can't be parsed as JSON. This is most often a human-friendly greeting to indicate that a component has connected to the WebSocket server.	The message is printed to the console for diagnostic purposes.
'setRules'	A 'write' request by the rule editor; The administrator has changed the rules in the rule editor (either by changing the activation state, deleting/creating a rule, or editing a rule and pressing save).	The 'Rules' table is reset and receives the new rules. A confirmation is sent to the rule editor with a copy of the rules as a JSON message of type 'ruleSetConfirmation'.
'getRules'	A 'read' request by the rule editor; The editor is being loaded for the first time and the currently valid set of rules is being sought.	The contents of the 'Rules' table are read and passed to the rule editor as a JSON message of type 'rules'.
'start'	The primary initialization command; The administrator has pressed the "Start Session" button in the Rule Editor to initiate the watching of events and execution of changes.	The DatabaseWriter subscribes to changes in documents matching the conditions specified in each rule. When a changefeed yields new data, a callback function prints information about the detected event to the console and broadcasts a message of type 'situationDetection' to all WebSocket clients.
'action Execution Request'	The EvaluationAndAdaptation Component was notified that a condition was triggered, connected to the database to determine which rule contained the condition, fetched all of the resulting actions, and then communicated a message of this type to the DatabaseWriter.	The DatabaseWriter broadcasts the adaptation request to all connected clients, including the web application.
'prompt Response'	The frontend web application previously prompted the user for input as a result of a 'prompt' command. The DatabaseWriter is now receiving the user's response to the query.	The user's answer is printed to the console. Future versions of the application could log the user's input to a log file or database.

Table 2: A Selection of Messages Received by the Database Writer

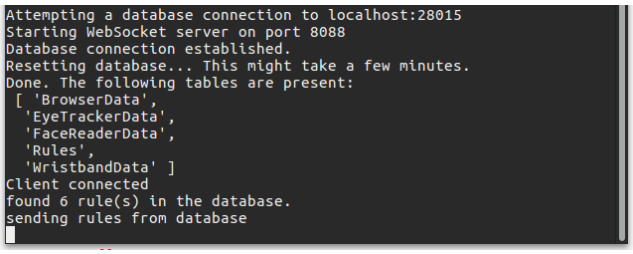

Figure 11: the Database Writer's starting output

The second responsibility of the Database Writer is to ferry incoming sensor data to the database. By adding an abstraction layer, instead of letting the RecordingComponent and the frontend application send data to the database directly, the system can be arranged more flexibly. Additionally, a comprehensive diagnostic interface is established. For every sensor, the Database Writer also maintains state information about the timestamp of the last message to have arrived. If no data is received for 6000 milliseconds, a warning is printed to the console. The console also prints an update if a data source starts or resumes streaming data.

The ferrying of data is accomplished by listening for certain incoming JSON-formatted WebSocket messages, processing their contents, and inserting them into the database. In order to guarantee the existence of all relevant tables, a reset procedure takes place when the script is started, provided a database connection has been established. This procedure accomplishes the following:

1. If the 'sitadapt' database is missing, it is created.
2. If the 'Rules' table in the database is missing, it is created and is filled with a collection of default rules (DatabaseWriter/defaulRules.js).
3. The tables 'FaceReaderData', 'EyeTrackerData', 'WristbandData' and 'BrowserData' are cleared of entries or created.

Console output from the reset procedure can be seen in Figure 11.

3.2.2 The Situation Analytics Component

The Situation Analytics Component is part of the same JavaScript (NodeJS) module that hosts the Database Writer. It is responsible for interfacing with the Rule Editor (which lets application administrators define the situation patterns that are active in the system) to determine what the relevant complex events of the system are, and then listening for those events to take place. In other words, it monitors sensor data for any administrator-defined situations, and then notifies the Evaluation and Adaptation component about any such occurrences. Listening to events is accomplished by constructing and subscribing to RethinkDB's changefeeds in accordance with the conditions defined by the system administrator in the rule editor. This step only takes place when an adaptation session is deliberately started by the administrator, as changefeeds cannot be unsubscribed from, whereas the rules in the editor can be changed continuously. Figure 12 shows the Database Writer's console output for the subscription of changefeeds.

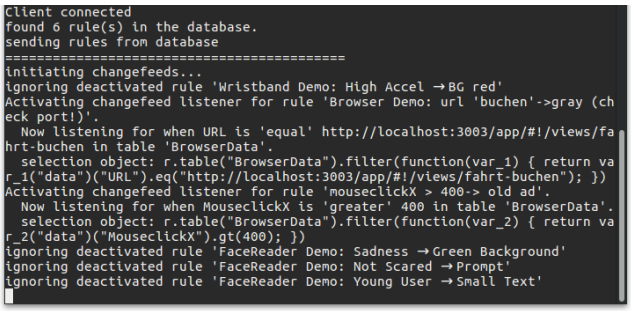

Figure 12: the Database Writer's output when a session is started

Changefeed Subscription

As outlined in section 2.2.5, RethinkDB's changefeeds allow developers to pass an arbitrarily complex query to the database along with a callback function that is called when the results of the query change. SitAdapt 2.0 subscribes to a changefeed for every situational pattern, with the conditions of the pattern being converted into the equivalent of an if-then function. The following code snippet, modified for clarity, illustrates the programmatic construction of a RethinkDB filter function used in changefeed subscription:

```
let selection = r.table(table).filter(function(item) {
  let left = item("data")(condition.signal);
  let right = condition.comparate;
  switch (condition.operator) {
    case 'greater':
      return left.gt(right);
    case "less":
      return left.lt(right);
    case "equal":
      return left.eq(right);
    case "not_equal":
      return left.ne(right);
  }
});

selection.changes().run(...) {
  cursor.each(function() {
    wss.clients.forEach(function (client) {
      client.send(JSON.stringify(
        {
          type:"situationDetection",
          data:{"condition":condition}
        }));
    });
  });
});
```

Of course, situations can have arbitrarily many conditions and are usually chained together with a boolean operation. Note that the current implementation naively regards any and all changes to the filtered row as an event trigger, and that the ANY or ALL selection in the Rule Editor does not have an effect on the creation of the changefeed filter function.

3.2.3 The Rule Editor

The Rule Editor is the second component to be executed when starting the system. This graphical tool allows application administrators to input conditions and responses (“situations” and “actions”) comfortably without programming XML or JSON directly. It was developed using Facebook’s ’create-react-app’ development package, which features a variety of configuration and execution scripts for developing a ReactJS web app.
The start script, for example, compiles the JSX-formatted React component description into HTML and CSS, then starts a webserver and opens the Rule Editor in a new browser tab.

Using the Rule Editor, a system administrator can activate or deactivate, add or delete, and modify existing rules. When a rule is changed, the full set of rules is sent to the DatabaseWriter.

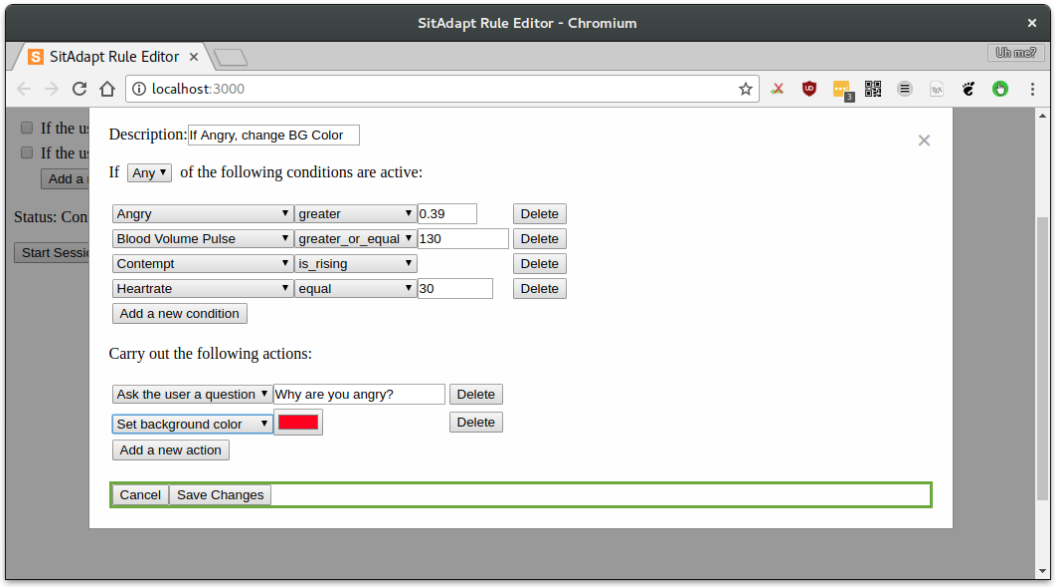

Figure 13: Screenshot of the Rule Editor in action

The development of the Rule Editor took place in two stages. At first, the editor was designed using normal (“vanilla”) JavaScript incrementally. Despite having a working prototype of the rule editor at a relatively early stage of development (with the ability to add and remove various rules and situations (even with category selection)), an influx of ideas for the expansion of the rule definitions led to the necessity of moving from a singular monolithic architecture to a modular design.

Despite a brief foray into TypeScript, a JavaScript flavor with static typing and a stronger support for object-oriented programming, Facebook’s React proved to be the framework of choice for the development of a flexible, modularly built user interface. As detailed in section 2.2.2, React’s data-binding and conditional rendering makes it possible for editor components such as the selection of input parameters to match the corresponding comparator. For example, selecting ”Ethnicity” will make the comparison value’s drop-down menu load possible values for that selection. Alternatively, if the administrator selects ”equal” as the comparison to an input such as ”URL” data, a textbox will spawn.

Modifications, deletions and additions to the situational patterns are communicated to the Database Writer via WebSocket messages.

Hierarchy of Components

The editor features a hierarchy of container components. In ”RuleEditor/public”, the index.html file’s element is filled by React using the contents of the index.js file in ”RuleEditor/src/”. There, a RuleManager component is the top-level element, containing a table of rules (with the ability to add, edit and delete rules). When the application’s ”state.modalIsOpen” variable evaluates to true, a modal dialog is also spawned, which enables editing of individual rules in a floating popup-like element. This modal (called ”MyModal” in index.js) is made up of a ”Description” component (to change the name of the rule), an ”AnyOrAll” component (should actions be triggered if ANY or if ALL of the conditions are true? - note this component presently has no effect on the system), a table of configurable ”Condition” components, and a table of configurable ”Action” components, as well as Save and Cancel buttons. Figure 14 shows the editor’s dropdown options for the condition’s signal and the rule’s actions.

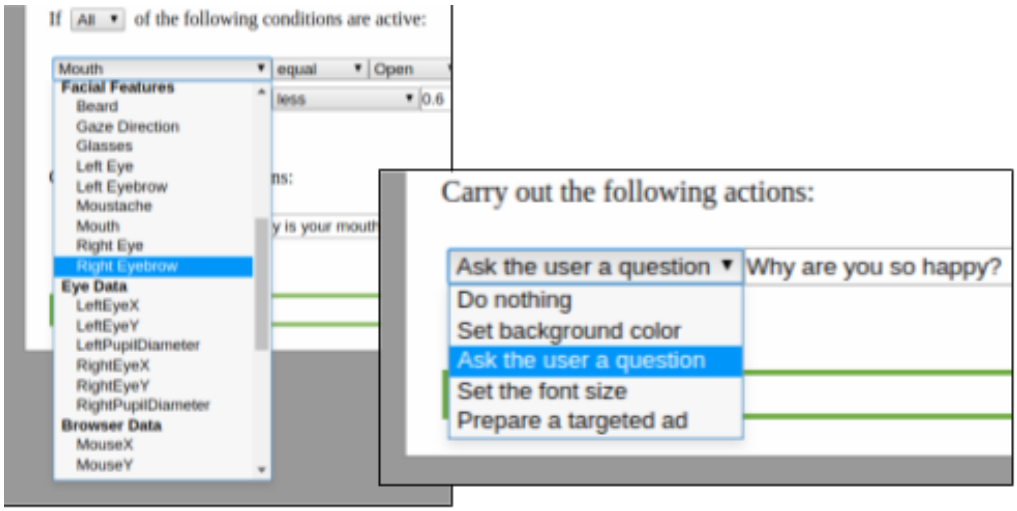

Figure 14: Screenshots of the Rule Editor’s Conditions and Actions

The following code listing exemplifies a portion of the RuleEditor (with some omissions for clarity), specifically the Action component. The Action component is ”invoked” from its parent container <MyModal> by including it as the html element <Action action={action} onChange={...} />, passing along an ”action” object (which contains a ”command” (e.g. ”setBackgroundColor”) and a parameter (e.g. ”#FF0000” for the color red). Additionally, a changeHandler is declared (called from the Action component via ”this.props.onChange(...)”). Inside the Action component, the render method tells React how to turn the Action component into an HTML element. In this example, the Action component is rendered as a div element containing a selection element for the action’s command and a different custom component for the action’s parameter. ”sitadaptActionsMapped” (in /DatabaseWriter/) is a file containing all of the system’s possible actions and their associated input types, as well as validation information. See Figure 13 for a screenshot of the editor in action. Three instances of the Action component are visible following the ”Carry out the following actions” text.

```
class Action extends React.Component {
  /* ... */
  commandChange(newCommand) {
    let newAction = this.props.action;
    newAction.command = newCommand;
    if (sitadaptActionsMapped[this.props.action.command]
    .parameter !== undefined) {
        newAction.argument =
            sitadaptActionsMapped[this.props.action.command]
            .parameter.value;
    }
    this.props.onChange(newAction)
  }

  parameterChange(newParameter) {
    let newAction = this.props.action;
    newAction.argument = newParameter;
    this.props.onChange(newAction)
  }

  render() {
    return (
      <div className="modalAction">
        <select type="select"
        value={sitadaptActionsMapped[this.props.action.command].id}
        onChange={(e) => this.commandChange(e.target.value)}>
        {sitadaptActions.map(action =>
            <option key={action.id} value={action.id} >
              {action.label}
            </option>
        )}
        </select>
        <ActionParameter action={this.props.action}
          onChange={
            (newParameter) => this.parameterChange(newParameter)
          }
        />
      </div>
    );
  }
}
```

Even though the package includes scripts to convert the project into statically hostable HTML, JS and CSS files, we chose to only use the development mode for maximum configurability.

3.2.4 The Recording Component

After the Database Writer and the Rule Editor, the third module to be started is the Recording Component, which passes input from the three hardware sensors to the Database Writer. After successfully connecting to the WebSocket server, the application connects and subscribes to incoming data of each hardware device individually. The EyeTrackerInitializer scans the network for attached EyeTrackers and connects to the first, and both the FaceReaderInitializer and WristbandInitializer attempt a connection to a specific port on the local machine for their corresponding devices.

In terms of subscribing to data, the .NET packages for the EyeTracker and FaceReader allow developers to easily attach functions to certain events caused by the devices (the most important ones

being the EyeTracker's 'GazeDataReceived' event and the FaceReader's 'ClassificationReceived' event). The Wristband's subscription process is slighty more complex, as there is no C#/.NET-integrated exception handling, and the subscription to every data channel has to be confirmed by waiting for an acknowledgement message by the wristband. Once the wristband and all of its data channels (acceleration, pulse, temperature, etc.) have been subscribed to, the WristbandInitializer calls the HandleResponseFromEmpaticaBLEServer function every time the socket connected to the the BLE Server receives some data. This function then processes the message.

As soon as any event handler for the wristband, EyeTracker or FaceReader has processed a message (in accordance with the steps outlined in following paragraphs) the data is sent to the Database Writer.

Language Choice

Due to the fact that the system's predecessor was developed as a console-based C# application, and because the only common language of the hardware API of all three hardware sensors was C#, it quickly became apparent that C# was the only sensible choice for coding the Recording Component's framework.

Methodology

The operating procedure of the Recording Component is as follows:

1. Connect to the Database Writer
2. Connect to each sensor via their respective API or SDK
3. Subscribe to incoming sensor data
4. When new data arrives, check its validity
5. If the data is valid, repackage it as JSON data
6. Finally, send the data to the database

Each sensor's API is slightly different, so steps 2, 3, and 4 vary for each sensor, as detailed in the folowing pages.

A planned addition to the system is the implementation of a baseline system that will measure the user in a resting state and use this default value when interpreting physiological and emotional data as being higher or lower than the ”average”.

The following section explains how the different hardware sensors are handled by the Recording Component in more detail.

Noldus FaceReader

SitAdapt 2.0 receives information about the user's facial characteristics via the FaceReader 7 Application Programming Interface (API). This data includes quantitative data for the user's emotional state (the degree to which they are angry, sad, surprised, etc) and his or her physical features (eyebrows raised/lowered, ethnicity, facial hair, etc.).

To start recording facial information, SitAdapt 2.0 (specifically, the Recording Component) connects

to a locally running instance of Noldus FaceReader and instructs it to start a new analysis session. The system then attaches an EventHandler for the ClassificationReceived event. When new data is received in the form of a ”detailed” classification log, each log entry’s key and value pairs are extracted and added to one of two temporary dictionaries: either an ”emotionDataPacket” or a ”subjectInfoPacket”, respective of the classification type. Additionally, the system’s current time is assessed and added as a timestamp to both dictionaries. The dictionaries are then serialized to a JSON string and pushed to the Database Writer. To simplify the serialization process, emotional data is converted to a so-called SitadaptQuantitativeDataPacket (a String to Double mapping, as seen in the code snippet below), as all of the attributes have numerical values, while the subject information data is converted to a SitadaptQualitativeDataPacket (a String to String mapping).

```
public class SitadaptQuantitativeDataPacket
{
    public string type { get; set; } //e.g. "emotionData"
    public string source { get; set; } //e.g. "FaceReader"
    //timestamp in milliseconds since epoch
    public long time { get; set; }
    public Dictionary<string, double> data { get; set; }

    public SitadaptQuantitativeDataPacket()
    {
        data = new Dictionary<string, double>();
    }
}
```

Unresolved issues with the FaceReader API include the reliance on a manual setup of the recording environment each time the system is restarted, as well as discrepancies regarding the accuracy of timestamps.

The Tobii EyeTracker

The Tobii Pro SDK is used to determine the screen-based coordinates of where the user is looking, employing a Tobii X2-60 EyeTracker. Other available data includes the user’s pupil size and the validity of each observation. These measurements are recorded on a per-eye basis. In the future, these coordinates could be used in conjunction with a web browser’s ”document.elementsFromPoint(x, y);” function, which returns an array of all document object model (DOM) elements at the specified location on the page, in order to determine what item the user is looking at.

In contrast to the FaceReader API, a connection with the EyeTracker is established by scanning the network for a list of all available EyeTrackers and connecting to the intended device. In our lab setup, this is always the first result. Once connected, an EventHandler for the GazeDataReceived is attached, which processes incoming data in a similar manner as the FaceReader. One small difference with the EyeTracker data is that the incoming data will only be processed if each eye’s GazePoint data is listed as ”Valid”. Next, the data is transformed into a SitadaptQuantitativeDataPacket just like the FaceReader emotion data.

Calibration and Synchronization of the Tobii EyeTracker

While the Tobii Pro SDK does include a GetSystemTimeStamp() function as well as a DeviceTimeStamp attribute for each GazeDataReceived event, the values of these items seem to correspond to the local computer's uptime rather than the start of the recording. Thus, a local system timestamp is used here as well, in lieu of a better synchronization method.
It also bears mentioning that the EyeTracker device must almost always be recalibrated when a new session is started, as the user's position in front of the sensor may have shifted slightly. While SitAdapt 1.0 featured its own built-in calibration method, SitAdapt 2.0 requires the system administrator to use the external EyeTracker Manager application's calibration feature.

Empatica Wristband

The Empatica Wristband is the newest edition to the SitAdapt system, offering real-time physiological data in the form of skin conductivity, pulse, temperature, and 3D acceleration monitoring. A bluetooth connection is established with the wristband using a BlueGiga BLED112 USB dongle, an instance of Empatica's Bluetooth Low Energy Server and a modified version of the Empatica BLE Client called the "WristbandInitializer". The latter is the part of the Recording Component responsible for establishing a connection with the wristband. If the connection is established successfully, the component repeatedly attempts to subscribe to each available data stream until a subscription confirmation is received for each stream, after which the data starts transmitting.
Due to inherent flaws in the wireless transmission of data, a lack of an event-based SDK, and inconsistencies in Empatica's output format (see [55]), the processing of the wristband data involves several syntactic and semantic checks, as detailed in the following steps, for the analysis of a single line of data:

1. Check that the line starts with the correct header ("E4_").
2. Check that the indicated data stream matches the expected stream title (e.g. "E4_acc").
3. Check that the timestamp does not lie in the past. Note that the timestamp is analyzed for each stream individually, meaning e.g. that the acceleration stream's last timestamps might be 1001, 1002 and 1003 while the temperature stream's last timestamps, which arrive AFTER the acceleration data, might be 995, 996 and 997, and would still be valid. Out-of-order data is only thrown out if it arrives after data from the same stream. Of course timestamp checking is crucial to ensure that the system only has to process fresh data instead of potentially reorganizing data structures and re-processing old situations to deal with new data. Without the aforementioned built-in tolerance, some streams would have to be disqualified in their entirety.
4. Check that the data has the right number of parameters. Acceleration has an X, Y, Z and a timestamp component, for example.
5. Check the line's overall length. Two or more pieces of data might have become part of the same line due to the omittance of a newline character. Alternatively, part of the message may have been cut off.
6. Extract the data and check that each value is in the accepted range. An overview of expected value ranges is available in section 7.2.

7. If the data is valid, repackage it as a SitadaptQuantitativeDataPacket (see Listing 3.2.4) and, together with the current timestamp, serialize it to JSON and send it to the Database Writer.

3.2.5 The Evaluation and Adaptation Component

The fourth module that is to be activated when running the system is the Evaluation and Adaptation Component, responsible for carrying out actions in response to rules defined by the administrator. When a callback function is triggered by Situation Analytics Component's changefeeds, a notification is broadcast to the Evaluation and Adaptation Component containing information about the signal, operator and comparate that triggered the callback. This information is used to find all rules that contain this condition and retrieve all of their actions. These actions, usually in the form of a command (”setBackgroundColor”) and an argument (”#888888”) are passed back to the WebSocket server as an ”actionExecutionRequest”, which causes the message to be broadcast to all attached clients, including the WebSocket client of the frontend web application, which carries out the change.

3.2.6 The Frontend Application

The fifth module to be started is a server hosting the frontend web application. While the web browser is one of the more important ”sensors” of the entire system, using the .NET Recording Component to monitor mouse input and URL changes would require extra steps and require browser-specific process monitoring. Instead, we elected to add a small script to the web application (or to the browser) that connects to the database and transmits data by itself. The script listens to mouse-click and move events, as well as monitoring the site's current URL. Any mouse inputs or navigation attempts by the user are immediately communicated to the Database Writer.

3.2.7 Live Observer

As part of the observation component, a browser-based monitoring tool was devised to show several pieces of information while a session is being performed. To begin with, a window showing the user's current view would be visible, possibly followed by incoming data. Most importantly, the system's detected situations and resulting actions would be clearly visible. A mockup is shown in figure 15. While the component was not fully realized (and therefore removed) due to time restrictions, the most promising implementation details will be described here as a jumping off point for developers planning to add this feature.

Figure 15: An early mockup of the live observer

Broadcasting What the User Sees

Two primary options present themselves for broadcasting the user's browser session: Either an HTML DOM (Document Object Model)-based mutation listener approach (which would be able to accurately model the contents of the web browser in a very bandwidth-effective manner, however without the ability to see the user's cursor), or a live video stream of the user's desktop.

While attractive in terms of its promises, the mutation-based approach offers little in the way of community support or widespread adoption, with little in the way of up-to-date technical instructions. One potentially helpful article from 2012 is [56].

Live Video Transmission

Live video streaming, on the other hand, is a growing industry. While a wise data protection policy would prohibit streaming data from leaving the confines of the testing lab, free-to-use online services do provide attractive solutions for streaming video, in terms of their ease of setup. Two of the more prevalent video stream hosting platforms are Twitch.tv and YouTube. While Twitch is limited to gaming-related content (note that all streams are public), YouTube offers the possibility of creating private, invite-only streams. We briefly evaluated YouTube as a streaming middleware, however voted against it when it became apparent that it was not possible to embed video streams in our application's page, which was our primary usage goal.

Next, we evaluated software for streaming a user's desktop to computers in a local area network. The best software for this purpose seemed to be either "OBS" - Open Broadcaster Software (note that this requires setting up a Real-Time Messaging Protocol (RTMP) broadcasting middleware such as NGinx), or, as a single-tool alternative, the popular media player VLC.

The RTMP solution would entail installing NGinx (note that Gryphon is the last version to offer free RTMP streaming support) and configuring it in accordance with [57] as seen in figure 16. An RTMP-capable video decoding library would then be necessary to view the video stream in a browser, for example video.js.

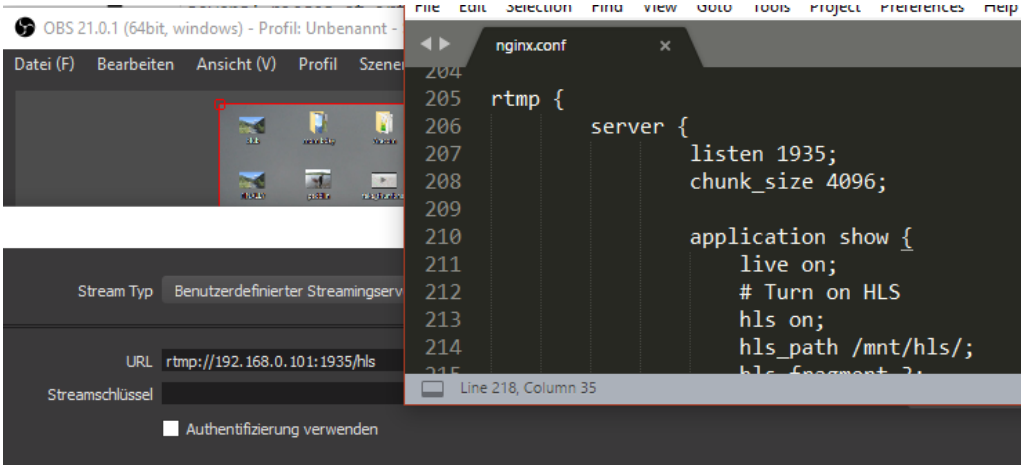

Figure 16: RTMP stream settings inside OBS; NGinx configuration on the right

The other viable live streaming solution was VideoLAN's VLC media player. A four-step streaming setup is shown in figure 17 and explained here.

1. From the File menu, select Stream and choose the desktop as the recording device. A higher framerate than the default (1 fps) is recommended.
2. Confirm the selection of the desktop ("screen://").
3. Add a new HTTP stream with a custom port and URL.
4. Choose Theora + Vorbis as the video and audio streaming format for maximum compatibility.

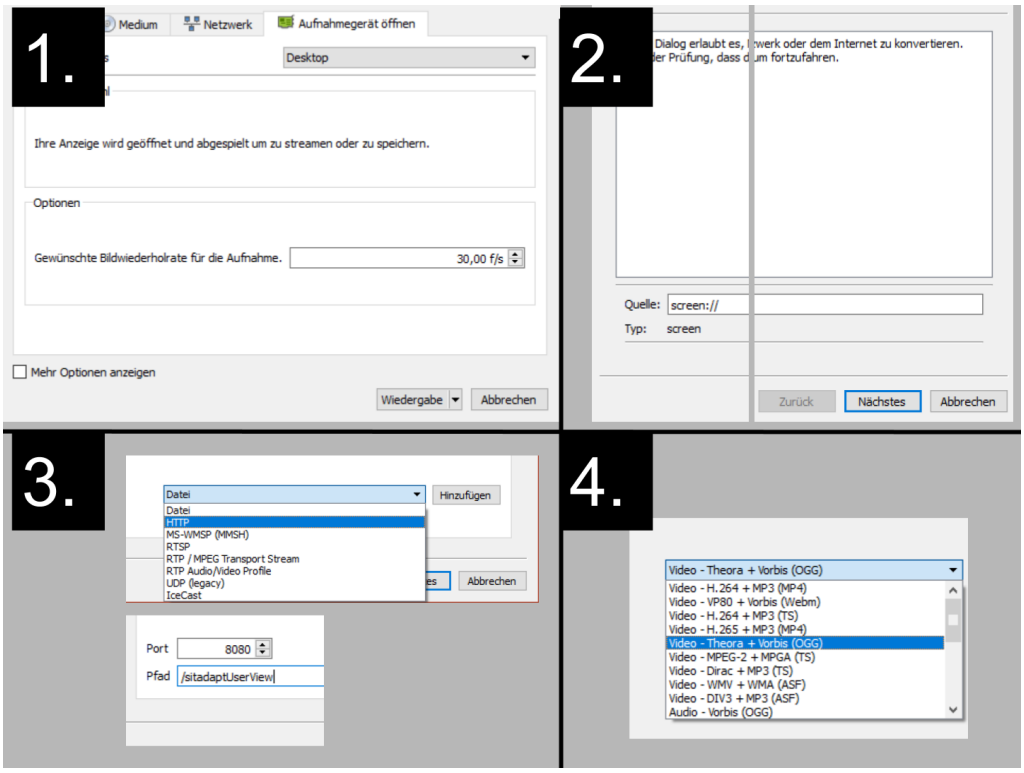

Figure 17: Configuration instructions for streaming desktop video with VLC over HTTP

For non-RTMP streams, a simple video tag would suffice to contain the video stream:

```
<video width="700" src="http://127.0.0.1:8084" autoplay
  type="video/ogg; codecs=theora" controls="controls">
</video>
```

It should be noted, however, that experimental results with a a VLC-based video stream showed an 8000 millisecond delay between recording the screen and displaying it to a web browser on the same computer. In order to show video, data visualizations, recognized events and triggered adaptations in a synchronized manner, the latter would all have to be delayed by 8000 milliseconds as well. These drawbacks compromise the intended use as a **live** observer.

In terms of showing data visualizations, chart libraries such as HighCharts, Charts.JS, Google Charts and others might proof valuable to show live incoming data. However, as shown in figure 18, various non-traditional data visualizations such as the position of the user's eyes (or mouse cursor), the size of user's pupils, shifting emotional data and others would put a strain on the integration of visualization libraries in a single, performant web interface. The two primary implementation techniques suggested to overcome this challenge are either designing a custom HTML Canvas rendering library specifically for SitAdapt, or using an advanced, low-level visualization library such as D3.JS.

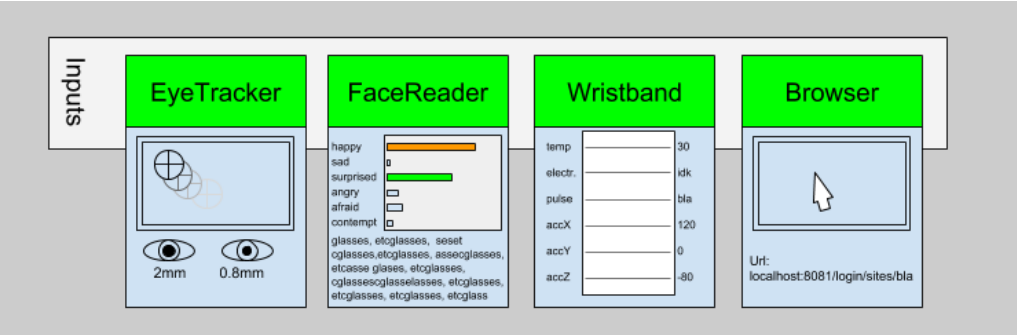

Figure 18: A mockup of the various data visualization elements for the live observer

3.2.8 Offline Observer

An offline observation would ideally make use of the same interface as the live observer while possibly enjoying a more manageable implementation of the screen and user recording playback. The component was not realized due to time restrictions.

Screen Recording

As part of the observation component of the system, the ability to accurately monitor and recreate the user's experience was considered to be of particular interest. In terms of implementation, one solution involved recording a video of the user's screen for later use, unfortunately ruling out live video playback transmission. For this approach, we made use of Mathew Sachin's Captura recording program [58] in conjunction with SharpAvi to record a high resolution MotionJPEG video at 10 frames per second. A comfortable tradeoff between filesize and framerate was not able to be established, however. Efforts to employ a low filesize, live-transmission-capable monitoring of the rendered website were still being pursued when the component was excluded from the system due to a lack of time. An alternative recording solution could involve embedding an element on a website tracking all mutations to the Document Object Model of the rendered HTML site, then transmitting these changes either to the database or directly to an observer instance.

User Recording

In addition to recording the user's screen, a recording of the user's face could also prove to be of interest to situation pattern evaluation efforts. If an offline replay (that is, a post-session) analysis were to suffice, one could simple view the recorded videos of Noldus FaceReader after finishing a recording. However, in order to enable the live observation of the user, a second webcam is required to record and broadcast the user's facial expressions. On the other hand, a lengthy manual examination of responses to application elements may not be desirable in the face of facial analysis software.

3.3 Interacting with the Frontend Web Application

3.3.1 Adaptation Evaluation

In the process of a creating an example frontend application as a part of SitAdapt 2.0, several possible adaptations were discussed, as seen in the following figure. The effort for the implementation of each adaptation in the frontend is colored red for difficult, yellow for moderate, and green for relatively simple.

To avoid overburdening the implementation of the frontend, the scope of the presentable examples of SitAdapt 2.0 was limited to four triggered adaptations: Changing the color of the page, changing the text size, spawning a feedback form, and updating the application state to show a specific advertisement when the user clicks on certain elements.

3.3.2 Adaptation Implementation

SitAdapt 2.0's example frontend web application was designed and implemented by Lionnelle Biawan Yameni as part of her Bachelor's thesis. An executable version is included in the folder "Kaffee"

for the purpose of demonstrating SitAdapt 2.0's capabilities. In the frontend application, each adaptation is activated in the frontend via a function embedded in each site's controller. A color change is performed by modifying the site's document.body.style. backgroundColor, a text property change is performed by modifying an individual element's style property (i.e. the fontSize subproperty) and a feedback prompt is spawned by simply calling the JavaScript function "prompt()". Note that the web application must be in focus for the prompt to appear, due to the spam-defense mechanisms of modern browsers.

Triggering

When a changefeed is triggered, a WebSocket message for every action is sent to the frontend application. A monitoring function in the site's controller listens for incoming WebSocket messages and calls the corresponding functions with the transmitted arguments.

E-Commerce Example

This section seeks to demonstrate a possible use case of the SitAdapt system. Designed alongside SitAdapt 2.0, Biawan Yameni's mock travel booking web application highlights some of the system's capabilities. The application features elements typical for e-commerce applications, including the ability to enter query parameters, viewing and selecting query results, viewing product details, registering as a new user and logging in, modifying a selected product, and viewing a summary of all entered data before committing to a purchase. The following example assumes that a system administrator has setup specific triggers and actions via the Rule Editor.

Having the ability to collect information about a user's physical and physiological properties allows application designers to offer products and product enrichment opportunities ("extras") that center on the user's immediate and/or long-term needs, benefiting producers and consumers. In the field of air travel, for instance, knowledge of a user's demographic and his or her current disposition can influence the kinds of seat and airline upgrade opportunities offered to the user.

In our first example, the system can recognize fear, anger, or a higher pulse while a user is in the process of choosing a flight. In response, it offers an effective remedy to combat the customer's fear of flying and/or other negative flight-associated emotions.

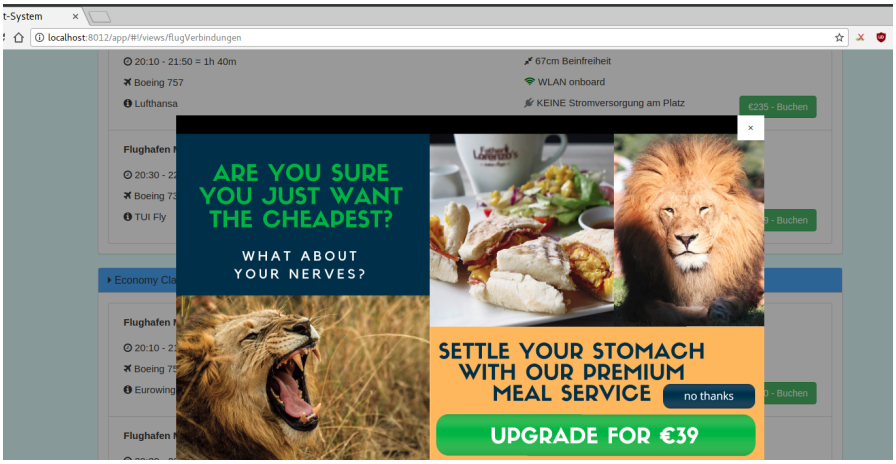

Figure 19: The system can offer a remedy for those nervous about flying

In our second example, the system uses a demographic-focused approach to determine positive flight-related desires and presents an option to fulfill these desires to the user. A user between the ages of 15 and 40 may be presented with an option to purchase WiFi access, while a user between the ages of 65 and 95 may be presented with seat upgrade opportunity to increase their level of comfort throughout the flight.

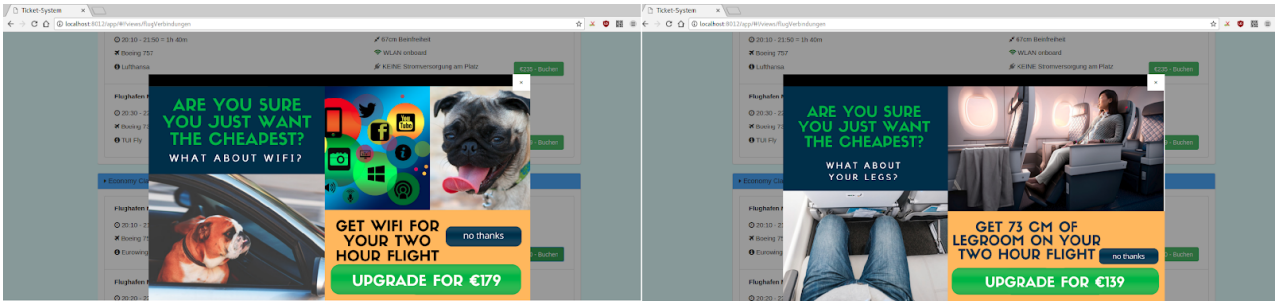

Figure 20: The system can prepare advertisements for various inferred needs

In theory, additional sensor information could also be used to make predictions about the likelihood of users taking advantage of these product enrichment opportunities. Eye tracking and browser-based mouse tracking data can offer information about a customer's flexibility as they consider or reconsider their budgetary restrictions when looking at the different opportunities.

Of course, the SitAdapt system's capabilities are not limited to targeted advertising. Application developers could also use our system to change color schemes to appeal to the user's current mood, change the text size to suit demographic-inferred needs, record specific diagnostic information and offer help (or ask for specific feedback) when a user becomes frustrated with a certain feature.

3.4 Walkthrough of a SitAdapt 2.0 Session

This section describes a typical walkthrough of a session in SitAdapt 2.0.

1. The administrator connects and calibrates some hardware. This step is optional.
2. The administrator starts an instance of RethinkDB.
3. The administrator starts the Database Writer.
   a) The DatabaseWriter connects to the RethinkDB instance and sets up the database.
      i. A connection to the database is established.
      ii. If the 'sitadapt' database is missing, it is created.
      iii. If the 'Rules' table in the database is missing, it is created and is filled with a collection of default rules.
      iv. The tables 'FaceReaderData', 'EyeTrackerData', 'WristbandData' and 'BrowserData' are created or reset.
   b) The DatabaseWriter starts the WebSocket server and starts listening for messages.
4. The administrator starts the Rule Editor.

a) The React App's start script compiles the website's static files and hosts them via a local webserver. A browser tab opens displaying the Rule Editor web application.
b) The Rule Editor requests rules from the Database Writer.
c) The Database Writer queries the database for the current rules and sends them to the web application.
d) The Rule Editor renders the currently active rules.

5. The administrator decides to modify some rules via the Rule Editor and clicks the Edit button.
   a) The application's 'modalIsOpen' property is set to true and the modal editor is rendered.
   b) The administrator decides to add a new condition and clicks on 'Add a new condition'.
   c) A new condition with default properties is added to the application state and rendered.
   d) The administrator clicks on 'Save Changes' and the modal dialog is hidden.
   e) The rule's state in the Rule Manager is overwritten by the Modal Editor's state, followed by the new rule set being transmitted to the Database Writer.
   f) The Database Writer records the new rules and sends back a confirmation.
   g) The Rule Editor checks the confirmation message and displays the success message "Connected and Synchronized"
6. The administrator starts the Recording Component. This step is only required if data from hardware components is to added to the system.
   a) The Recording Component establishes a connection to the WebSocket server.
   b) Each sensor's initializer connects to the device and subscribes to data incoming data events, as detailed in Section 3.2.4.
   c) When data arrives, it is validated, repackaged as a SitadaptQuantitativeDataPacket or SitadaptQualitativeDataPacket, converted to JSON , and sent to the Database Writer.
7. The administrator starts the EvaluationAndAdaptation Component.
   a) The component establishes a connection to the RethinkDB instance and the WebSocket server.
   b) The component starts listening for messages of type 'situationDetection'.
8. The administrator clicks the "Start Session" button in the Rule Editor.
   a) A message of type 'start' is sent to the Database Writer.
   b) For every condition of every rule in the 'Rules' table, the Database Writer subscribes to a changefeed.
      i. The corresponding database table for the condition's signal (left operand) is determined, e.g. 'FaceReaderData' for the signal 'gender'.
      ii. A filter function is constructed using the condition's signal, the condition's operator (e.g. 'not_equal'), and the condition's comparator (e.g. 'Male').

iii. The Database Writer subscribes to changes in the ascertained table for queries that match the filter function.

iv. Upon detection, a message of type ’situationDetection’ is sent to the EvaluationAndAdaptation Component.

9. The administrator starts a server hosting the frontend web application. A test subject can now begin browsing the website.

4 Evaluation and Conclusion

4.1 Fulfillment of Intent

The submitted system, SitAdapt 2.0, accomplishes most if not all of its intended goals, offering application developers a semi-automated situation-recognition and UI optimization engine. As an upgrade to SitAdapt 1.0, the system can now process data from a Tobii EyeTracker, Noldus FaceReader, a web browser and an Empatica E4 Wristband. Additionally, data is now logged uniformly to a database. Furthermore, administrators can now comfortably define an arbitrary set of physiological conditions, which are evaluated and responded to in real-time via a multitude of interface adaptations triggered in a frontend application. As such, the project can be considered a success.

4.2 Concluding Remarks

Innovation for the sake of innovation has the potential of causing harm and unhealthy rifts to society. SitAdapt 2.0 may bring with it opportunities for good, with the ability to change all kinds of user interfaces for the better in terms of usability. Although our system doesn't to use input data to track users over a period of time, it does require users to submit data in a way that invades their privacy. One of SitAdapt 2.0's example applications discriminates openly regarding a user's age, and influences their shopping behavior without any benefit to the user in terms of application usability. Additional ethical considerations are raised by the system's long-term storage of physiological data. One should therefore be careful about the motivations for employing such a system.

On the other hand, making computers more aware of a user's needs and impressions may bridge the gap between those well-versed in using such technology and those that have less experience. We postulate that by making applications smarter and more intuitive, technology becomes more accessible to all.

5 Future Work

5.1 Making SitAdapt Portable

5.1.1 Creating a Single Executable Program

SitAdapt 2.0 currently requires users to execute five different commands to start the system. Combining these scripts into a single application would reduce complexity, mitigate the prevalence of cross-component communication bugs and minimize the risk of human error, in addition to making it easier to publish and distribute the software. As is common for prototypical software, many parts of the application would benefit from being refactored.

5.1.2 Correlating Physiological Data with Browser Data

Another improvement to the system would be portability. Work is currently underway to move SitAdapt ”out of the lab” by finding correlations between data that is only accessible in a laboratory environment (pulse, pupil dilation, facial features) and data that is readily available to a standard website (input events and application metadata). Example findings might include matching mouse jitter to an increase in pulse, longer periods of time between scroll events to a decreased dilation of the eye, or the speed and frequency of backspace key usage to a user’s age or tiredness level. Future endeavours may be able to then use the resulting analysis and apply it to a new version of SitAdapt. Certain parts of SitAdapt 2.0’s architecture would need to be redesigned, however.

5.1.3 Using SitAdapt in Other Applications

In order for the system to be applicable to a variety of software and hardware applications, a specific set of requirements could be developed for frontend web applications that seek to use the system. These could even be made testable, meaning that the system could analyze a given application and give developers feedback about what data the frontend application is not yet disclosing, or what commands it is not handling. Alternatively, an application programming interface could be defined consisting of a set of data-collection and UI adaptation methods.

5.2 Adding Features to SitAdapt

5.2.1 More Contextual Information

In contrast to approaches such as focused pupillometry in [21], SitAdapt 2.0 features much more of a ”clunky” approach, in that the ability to collect information about an application’s context is limited to URL data received from the browser. In the current implementation, the provided metadata isn’t sufficient to make fine-grained sensor-infused decisions about the application state. In other words, it is difficult to procedurally connect input data (e.g. pupil dilation) to the steps taken by the user in the current part of the application.

SitAdapt 2.0 would be able to act in a more refined manner to a user’s context if application metadata was more abundant. For example, knowing which products the user has selected, or the number of times the user has glanced at certain panels would allow the system to more accurately

infer the user's individual priorities, in terms of price and feature selection, and customize future content to fit a customer's interests and budget.

5.2.2 Combination of Conditions

While the submitted system's Rule Editor lets administrators choose either a union or an intersection of conditions (Does just one condition have to be active to trigger a rule, or do all of them need to be active?), this selection **has no effect** on the system. Changefeeds are naively subscribed to for every condition of every active rule without taking account the ANY-or-ALL selector. In addition to misleading administrators about the configuration of the system, this oversight prevents SitAdapt 2.0 from being able to act on physiological data in a context-aware manner.

5.2.3 Dynamic Conditions

The present system only allows input data to be compared with static, hard-coded values. While user-specific calibration is possible for emotional data (via the configuration options of FaceReader 7.0's emotional classifier), there is no way to determine if the cardiological data being received by the wristband is higher or lower than a subject's average heartrate. By adding a baseline recording phase to the instantiation process of the system, physiological data can be recorded for every user in a relaxed state. The Rule Editor could then be expanded to allow system administrators to compare incoming input data with the user's baseline stats.

5.2.4 Custom Operators

Another sought-after addition to the application's rules would involve adding new comparison operators. In contrast to the existing operators (”equal”, ”greater than”, ”less than or equal to”, etc), these custom operators would not involve a comparison with another value. Instead, the operator would evaluate if data is changing in a certain manner. Specific proposals include an ”is_rising” and ”is_falling” operator, which would determine if numeric data has been monotonically changing by a certain percentage in a certain period of time.

5.3 Optimizing UI Adaptations

5.3.1 Consulting a Library of UI Changes

UI adaptation systems like SitAdapt would benefit from a large, detailed repository of situations and recommended UI optimizations for a given input of affect-related and physiological data. One of the problems with the implementation of such a collection might be the vagueness of the term ”optimization”. Some applications may seek to minimize frustration, aggression or stress, others might try to maximize enjoyability, happiness and positive surprise. Some might simply try to optimize profitability. And in any case, there will almost always be more than one way to change a user interface to achieve a certain goal for a given behavioral change. The PaMGIS framework [51] includes a repository of model-based UI design patterns (including usability evaluation), while the official SitAdapt architecture adds affect-related data to PaMGIS UI models at runtime. However, neither approach features a library of best practices for a given set of physiological data.

Device	Software
Eye Tracker	Tobii Eye Tracker Manager
E4 Wristband	Empatica BLE Server
Camera	Noldus FaceReader

Table 3: Hardware sensors and their software tools

5.3.2 Incorporating User Feedback

In order to evaluate the effectiveness of any UI adaptation, some kind of feedback system could be implemented. This could be designed in conjunction with the PaMGIS Pattern Specification Language, which stores evaluation data as part of each pattern's entry in a pattern repository. The feedback could be used either by application developers to optimize adaptation situations and rules manually, or by some kind of automated optimization agent that incrementally updates optimization rules to maximize certain metrics.

5.4 Making SitAdapt More Robust

5.4.1 Dealing With Data Outliers and Extremes

The current implementation of the system acts on instantaneous data events, meaning any new data values are assumed to be an accurate state of the user's current emotion. An algorithm that instead analyzes a window of time might be able to read the data more consistently. Averaging a user's ”angry” values for the last 1000 milliseconds, for example, can help mitigate unwarranted interfaces changes as a result of data fluctuation and extremes.

5.4.2 Employing More Reliable Hardware

SitAdapt 2.0 relies on three different hardware sensors that require a different first-time installation for each device. These sensors all depend on management software to function (or at least to be calibrated) as seen in Table 3. As the project envelopes a large chain of data processing steps (subscribing to data, validation of data, repackaging data as JSON, transmitting data, inserting data in database, analyzing data), every source of error or prolonged initialization procedure can cause grave disruptions to the use and development of the system. As a concrete example, the circumstance that the E4 Wristband sometimes takes longer than 10 minutes to connect to the BLE Server (without any kind of feedback in the meantime) can be a pain for application administrators and users. The software also has no notification when a device disconnects.

One solution to this dilemma would entail reevaluating the effectiveness of each hardware component and perhaps acquiring more dedicated, fault-tolerant hardware such as a wired pulse oximeter instead of the Empatica E4 Wristband. Of course, exchanging one multi-sensor interface for several single-sensor interfaces will increase the complexity of the entire application. However, the modular arrangement of inputs in SitAdapt 1.0 and SitAdapt 2.0 has demonstrated that complexity by itself does not pose a problem.

5.4.3 Synchronizing Data

In the area of data processing, SitAdapt 2.0 timestamps the data received from the hardware components in a very naive manner, namely by the time of their arrival. Future version of the system could attempt to synchronize data using information supplied by the individual sensors.
Even though some of the data available to the Recording Component does include timestamp information, this information was often poorly labelled or documented. Tobii Eyetracker's GetSystemTimeStamp() function, for example, doesn't return the current system time, but instead describes how long the computer has been running. The EyeTracker's other timing information, contained in the GazeDataReceived event's DeviceTimeStamp, was often nineteen or more minutes in the past. The Empatica E4's data streams also included timing information, however data would often arrive in an order that was inconsistent with the timestamp, calling the accuracy of the internal clock into question. As a solution, we elected to use C#'s DateTime.UtcNow function to generate a timestamp (milliseconds since the unix epoch) for all incoming sensor data.

5.4.4 Improving the Development Process

Automated Testing

One serious cause for concern during the development of the system was the reliance on manual testing after every change to verify the working state of the system. Writing automated tests that are executed after every change or commit (such as via GitLab's continuous integration feature, or some setup involving Jenkins and Gradle) would go a long way to ensuring that the system remains stable, in addition to making it easier to identify bugs. Automated tests that run in the background may also have the benefit of finding bugs associated with memory leaks, such as when a test is conducted for 30-60 minutes. Manual tests were executed for extended periods of time only rarely, making it all too likely for out-of-memory bugs to be present in the system.

Dummy Data

Another common source of problems during development was the unreliable data stream from hardware devices. The Wristband would regularly take a long time to connect to the computer's bluetooth dongle, the pulse tracking often gave out entirely, the CPU-intensive processing of Noldus FaceReader didn't always find the tester's face, and the EyeTracker frequently failed to find both of the tester's eyes. A generator of dummy data would allay these concerns and let future application developers focus on developing the software rather than spending precious time re-arranging and re-configuring hardware.

6 References

[1] *Internet Growth Statistics 1995 to 2017 - the Global Village Online*. http://www.internetworldstats.com/emarketing.htm. (Accessed on 15.01.2018).

[2] *Digital buyers worldwide 2021 | Statistic*. https://www.statista.com/statistics/251666/number-of-digital-buyers-worldwide/. (Accessed on 15.01.2018).

[3] *Mobile Phone Users and Penetration Worldwide, 2015-2020 (billions, % of population and % change) - eMarketer*. https://www.emarketer.com/Chart/Mobile-Phone-Users-Penetration-Worldwide-2015-2020-billions-of-population-change/196278. (Accessed on 16.01.2018).

[4] *GUIdebook > Articles > "Inventing the Lisa User Interface"*. https://guidebookgallery.org/articles/inventingthelisauserinterface. (Accessed on 16.01.2018).

[5] Sanim Rashid. *Entwicklung eines Prototypen zur User-Experience Optimierung auf der Basis von Emotions- und Blickanalyse im Bereich E-Commerce*. Apr. 2016.

[6] S Card, T Moran, and A Newell. *The Psychology of Human-Computer Interaction*. Hillside, New Jersey: Lawrence Erlbaum Associates, 1983. Chap. 1, p. 2.

[7] *Usable | Definition of Usable by Merriam-Webster*. https://www.merriam-webster.com/dictionary/usable. (Accessed on 02.02.2018).

[8] *What is Usability Engineering? | Interaction Design Foundation*. https://www.interaction-design.org/literature/topics/usability-engineering. (Accessed on 02.02.2018).

[9] Russel Beale. *Introduction to HCI*. https://www.cs.bham.ac.uk/~rxb/Teaching/HCI%20II/intro.html. (Accessed on 05.02.2018).

[10] Johannes Robier. "UX Redefined: Winning and Keeping Customers with Enhanced Usability and User Experience". In: Switzerland: Springer International Publishing, 2016. Chap. 1.3, pp. 13–14.

[11] Sean Landry. *What's the difference between UX and HCI? - Quora*. https://www.quora.com/Whats-the-difference-between-UX-and-HCI. (Accessed on 05.02.2018). July 2013.

[12] Stuart Reeves. *What Is the Relationship Between HCI Research and UX Practice? :: UXmatters*. https://www.uxmatters.com/mt/archives/2014/08/what-is-the-relationship-between-hci-research-and-ux-practice.php. (Accessed on 05.02.2018). Aug. 2014.

[13] Michael Swaine, William Pottenger, et al. *Computer*. https://www.britannica.com/technology/computer/History-of-computing. (Accessed on 12.02.2018). Jan. 2018.

[14] Jon R. Edwards. *A History of Early Computing at Princeton - Alan Turing Centennial*. https://www.princeton.edu/turing/alan/history-of-computing-at-p/. (Accessed on 18.03.2018). 2012.

[15] ScienCentral Inc. *The First Transistorized Computer*. http://www.pbs.org/transistor/background1/events/sscomputer.html. (Accessed on 18.03.018). 1999.

[16] Jakob Nielsen. *Usability Engineering*. San Diego, CA: Academic Press, 1993. Chap. 3, p. 67.

[17] Sari Kujala et al. “UX Curve: A method for evaluating long-term user experience”. In: *Interacting with Computers* 23.5 (2011), pp. 473–483. DOI: 10.1016/j.intcom.2011.06.005. eprint: /oup/backfile/content_public/journal/iwc/23/5/10.1016/j.intcom.2011.06.005/2/iwc23-0473.pdf. URL: +http://dx.doi.org/10.1016/j.intcom.2011.06.005.

[18] Rosalind Picard. *Affective Computing*. (Accessed on 06.02.2018). Cambridge, MA: MIT Media Laboratory Perceptual Computing Section, 1997.

[19] Dennis D. Fehrenbacher. “Affect Infusion and Detection through Faces in Computer-mediated Knowledge-sharing Decisions”. In: *Journal of the Association for Information Systems* 18.10 (2017).

[20] Joseph P Forgas. “Mood and judgment: The affect infusion model (AIM).” In: *Psychological Bulletin* 117.1 (1995), pp. 39–66.

[21] Dennis D. Fehrenbacher and Soussan Djamasbi. “Information systems and task demand: An exploratory pupillometry study of computerized decision making”. In: *Decision Support Systems* 97 (2017), pp. 1 –11. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2017.02.007. URL: http://www.sciencedirect.com/science/article/pii/S0167923617300246.

[22] David Luckham and Roy Schulte. *Event Processing Glossary – Version 2.0 – Real Time Intelligence & Complex Event Processing*. http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2/. (Accessed on 06.02.2018). July 2011.

[23] Opher Etzion. “Complex Event Processing”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 412–413. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_571. URL: https://doi.org/10.1007/978-0-387-39940-9_571.

[24] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, 2002, pp. 17–25. ISBN: 0201727897.

[25] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language*. New York: Oxford University Press, 1977, p. ix. URL: http://library.uniteddiversity.coop/Ecological_Building/A_Pattern_Language.pdf.

[26] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979. ISBN: 0195024028.

[27] Ahmed Seffah. *Patterns of HCI Design and HCI Design of Patterns*. Switzerland: Springer International Publishing, May 2015, pp. 1 –13. ISBN: 978-3-319-15686-6. URL: https://link.springer.com/book/10.1007/978-3-319-15687-3.

[28] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994, pp. 1 –8. ISBN: 0201633612.

[29] Ronald G. Ross. *Decision Rules vs. Behavioral Rules (Commentary)*. https://www.brcommunity.com/articles.php?id=b709. (Accessed on 09.02.2018). 2013.

[30] Anders Toxboe. *Design patterns*. http://ui-patterns.com/patterns. (Accessed on 09.02.2018).

[31] Ahmed Shamim Hassan. *Observer vs Pub-Sub pattern – Hacker Noon*. https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c. (Accessed on 14.02.2018). Oct. 2017.

[32] *User's Manual | Tobii X2-60 Eye Tracker*. (Accessed on 15.02.2018). Tobii Technology AB. Danderyd, Sweden, 2014.

[33] Robert J.K. Jacob and Keith S. Karn. "Commentary on Section 4 - Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises". In: *The Mind's Eye*. Ed. by J. Hyönä, R. Radach, and H. Deubel. Amsterdam: North-Holland, 2003, pp. 573 –605. ISBN: 978-0-444-51020-4. DOI: https://doi.org/10.1016/B978-044451020-4/50031-1. URL: https://www.sciencedirect.com/science/article/pii/B9780444510204500311.

[34] Paul Viola and Michael J. Jones. "Robust Real-Time Face Detection". In: *International Journal of Computer Vision* 57.2 (2004), pp. 137–154. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000013087.49260.fb. URL: https://doi.org/10.1023/B:VISI.0000013087.49260.fb.

[35] Leanne Joijens and Olga Krips. *FaceReader Methodology Note*. (Accessed on 07.02.2018). Wageningen, The Netherlands: Noldus Information Technology, 2018.

[36] Empatica S.R.L. *E4 Wristband User Manual*. https://empatica.app.box.com/v/E4-User-Manual. (Accessed on 05.02.2018). 2016.

[37] *Empatica BLE Server for Windows (Beta)*. http://developer.empatica.com/windows-ble-server.html. (Accessed on 22.01.2018).

[38] *Electrodermal activity - Wikipedia*. https://en.wikipedia.org/wiki/Electrodermal_activity. (Accessed on 15.02.2018).

[39] Jason J Braithwaite et al. *A Guide for Analysing Electrodermal Activity (EDA) & Skin Conductance Responses (SCRs) for Psychological Experiments*. https://www.birmingham.ac.uk/Documents/college-les/psych/saal/guide-electrodermal-activity.pdf. (Accessed on 15.02.2018). 2015.

[40] *Medical Definition of Parasympathetic nervous system*. https://www.medicinenet.com/script/main/art.asp?articlekey=4770. (Accessed on 15.02.2018).

[41] IETF | Internet Engineering Task Force. *Hypertext Transfer Protocol – HTTP/1.1*. https://tools.ietf.org/html/rfc2068. (Accessed on 08.02.2018). 1997.

[42] Tieme. *php - What are Long-Polling, Websockets, Server-Sent Events (SSE) and Comet? - Stack Overflow*. https://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet/12855533. (Accessed on 08.02.2018). 2012.

[43] Marc Gravell. *.Net 4.5 WebSocket Server Running on Windows 7? - Stack Overflow*. https://stackoverflow.com/questions/12073455/net-4-5-websocket-server-running-on-windows-7/12073593. (Accessed on 08.02.2018). 2012.

[44] Facebook Inc. *React - A JavaScript library for building user interfaces*. https://reactjs.org/. (Accessed on 13.02.2018). 2018.

[45] Jeremy Garcia. *Interview with Patrick Volkerding of Slackware*. https://www.linuxquestions.org/questions/interviews-28/interview-with-patrick-volkerding-of-slackware-949029/. (Accessed on 15.02.2018). June 2012.

[46] *RethinkDB Documentation - RethinkDB*. https://rethinkdb.com/docs/. (Accessed on 17.03.2018).

[47] Christian Herdin, Christian Märtin, and Peter Forbrig. “SitAdapt: An Architecture for Situation-Aware Runtime Adaptation of Interactive Systems”. In: *Human-Computer Interaction. User Interface Design, Development and Multimodality - 19th International Conference, HCI International 2017, Vancouver, BC, Canada, July 9-14, 2017, Proceedings, Part I*. 2017, pp. 447–455. DOI: 10.1007/978-3-319-58071-5_33. URL: https://doi.org/10.1007/978-3-319-58071-5_33.

[48] Gerrit Meixner, Gaëlle Calvary, and Joëlle Coutaz. *Introduction to Model-Based User Interfaces*. https://www.w3.org/TR/mbui-intro/. (Accessed on 12.02.2018). 2014.

[49] Marc Halbrügge. “Model-Based UI Development (MBUID)”. In: *Predicting User Performance and Errors: Automated Usability Evaluation Through Computational Introspection of Model-Based User Interfaces*. Cham: Springer International Publishing, 2018, pp. 19–22. ISBN: 978-3-319-60369-8. DOI: 10.1007/978-3-319-60369-8_3. URL: https://doi.org/10.1007/978-3-319-60369-8_3.

[50] Lionel Balme et al. “CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces”. In: *Ambient Intelligence*. Ed. by Panos Markopoulos et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 291–302. ISBN: 978-3-540-30473-9.

[51] Jürgen Engel and Christian Märtin. “PaMGIS: A Framework for Pattern-Based Modeling and Generation of Interactive Systems”. In: *Proceedings of the 13th International Conference on Human-Computer Interaction. Part I: New Trends*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 826–835. ISBN: 978-3-642-02573-0. DOI: 10.1007/978-3-642-02574-7_92. URL: http://dx.doi.org/10.1007/978-3-642-02574-7_92.

[52] Eric Céret, Sophie Dupuy-Chessa, and Julián Andrés Galindo. *Toward a UI Adaptation Approach Driven by User Emotions*. http://iihm.imag.fr/publs/2017/GalindoACHI2017.pdf. Mar. 2017.

[53] Fatma Nasoz. “Adaptive Intelligent User Interfaces with Emotion Recognition”. PhD thesis. M.S. University of Central Florida, 2004.

[54] Johannes Wagner et al. “The social signal interpretation (SSI) framework: multimodal signal processing and recognition in real-time”. In: *Proceedings of the 21st ACM international conference on Multimedia*. MM ’13. New York, NY, USA: ACM, 2013, pp. 831–834. ISBN: 978-1-4503-2404-5. DOI: 10.1145/2502081.2502223. URL: http://doi.acm.org/10.1145/2502081.2502223.

[55] *Issues · empatica/ble-client-windows*. https://github.com/empatica/ble-client-windows/issues. (Accessed on 17.03.2018).

[56] Eric Bidelman. *Screensharing a browser tab in HTML5? - HTML5 Rocks*. https://www.html5rocks.com/en/tutorials/streaming/screenshare/. (Accessed on 12.03.2018). Sept. 2012.

[57] 3DFun. *Question / Help - Live Streaming to a browser | Open Broadcaster Software*. https://obsproject.com/forum/threads/live-streaming-to-a-browser.14288/. (Accessed on 12.03.2018). May 2014.

[58] Mathew Sachin. *Captura*. https://mathewsachin.github.io/Captura/. (Accessed on 18.03.2018). 2018.

7 Appendix

7.1 List of Figures

All figures and images are the original work of the author of this thesis.

7.2 Input Data Description

EyeTracker Data

Attribute	Possible Values	Description
LeftPupilDiameter	between ca. 2.0 and 8.0	Describes the dilation of the subject's left pupil in mm.
RightPupilDiameter	between ca. 2.0 and 8.0	Describes the dilation of the subject's right pupil in mm.
LeftEyeX	between 0.0 and 1.0	Indicates the normalized x-coordinate of the subject's left eye. Normalized here means that the X,Y point at the top left of the monitor is defined as (0,0) and the bottom right point is defined as (1.0, 1.0).
LeftEyeY	between 0.0 and 1.0	Indicates the normalized y-coordinate of the subject's left eye.
RightEyeX	between 0.0 and 1.0	Indicates the normalized x-coordinate of the subject's right eye.
RightEyeY	between 0.0 and 1.0	Indicates the normalized y-coordinate of the subject's right eye.

FaceReader Data

Attribute	Possible Values	Description
Angry	between 0.0 and 1.0	Indicates to what degree the user appears angry.
Disgusted	between 0.0 and 1.0	Indicates to what degree the user appears disgusted.
Happy	between 0.0 and 1.0	Indicates to what degree the user appears happy.
Neutral	between 0.0 and 1.0	Indicates to what degree the user appears neutral.
Sad	between 0.0 and 1.0	Indicates to what degree the user appears sad.
Scared	between 0.0 and 1.0	Indicates to what degree the user appears scared.
Surprised	between 0.0 and 1.0	Indicates to what degree the user appears surprised.
Contempt	between 0.0 and 1.0	Indicates to what degree the user appears full of contempt. Contempt is defined as the feeling that a person or a thing is beneath consideration, worthless, or deserving scorn.
Valence	between -1.0 and 1.0	Describes how positive/negative an emotion is. -1 means very negative.
Arousal	between 0.0 and 1.0	Describes how strong the emotion is. A person that is yelling in anger, for example, has a higher arousal than a person who is only pulling their eyebrows together.
Quality	between 0.0 and 1.0	Roughly describes the quality of the observation, i.e. how well the subject's face could be analyzed. Being far away from the camera, poor lighting conditions or wearing big glasses can reduce the quality.
Age	”20-30”, ”35-45”, etc	Describes an interval for the estimated age of the user.
Beard	”None”, ”Some”, ”Full”	Describes the extent of the subject's lower facial hair.
Moustache	”None”, ”Some”, ”Full”	Describes the extent of the subject's upper facial hair.
Glasses	”Yes”, ”No”	Describes whether or not the user is wearing glasses.
Ethnicity	”Caucasian”, ”Eastern Asian”, ”South Asian”, ”African”, ”Other”	Describes the ethnicity of the subject based on visual appearance.
Gaze Direction	”Left”, ”Right”, ”Forward”	Describes in which direction the subject is looking.
Identity	”unknown person”, ”NO IDENTIFICATION”	Identifies the user based on the current Noldus FaceReader session's profiles.
Left Eye	”Open”, ”Closed”	Describes the state of the user's left eye.
Right Eye	”Open”, ”Closed”	Describes the state of the user's right eye.
Mouth	”Open”, ”Closed”	Describes the state of the user's mouth.
Left Eyebrow	”Raised”, ”Lowered”, ”Neutral”	Describes the state of the user's left eyebrow.
Right Eyebrow	”Raised”, ”Lowered”, ”Neutral”	Describes the state of the user's right eyebrow.

Wristband Data

Attribute	Value Range	Description
x	between -128 and 127	Acceleration value for the x axis, ”defined by the vector whose starting point is set to the center of the device and whose direction points towards the USB slot.” [37]
y	between -128 and 127	Acceleration value for the y axis, ”defined by the vector whose starting point is set to the center of the device and whose direction points towards the shorter strap.”
z	between -128 and 127	Acceleration value for the z axis, ”defined by the vector whose starting point is set to the center of the device and whose direction points towards the bottom of the device.”
bvp	between ca. -1000 and 1000	Blood Volume Pulse ”derived from the light absorbance of the arterial blood. The raw signal is filtered to remove movement artifacts.”
gsr	between ca. 0 and 6	Galvanic Skin Response, a measure of the skin’s electrical conductivity, ”expressed in microsiemens”
tmp	between ca. 15 and 40	Temperature in degrees Celsius. ”The value is derived from the optical temperature sensor placed on the wrist.”
ibi	between ca. 0.4 and 1.9	Interbeat Interval, ”the distance from the previous detected heartbeat in seconds.”
hr	between ca. 30 and 160	Heartrate in beats per minute
bat	either 0.01 or 0.1	Battery level (note this value appears to be bugged)

Browser Data

Attribute	Possible Values	Description
MouseX	between 0 and [screen-width]	The X position of the mouse in relation to the monitor for a mouseMove event.
MouseY	between 0 and [screen-height]	The Y position of the mouse in relation to the monitor for a mouseMove event.
MousePageX	between 0 and infinite	The X position of the mouse in relation to the page for a mouseMove event. This is effectively MouseX + the horizontal scroll distance.
MousePageY	between 0 and infinite	The Y position of the mouse in relation to the page for a mouseMove event. This is effectively MouseY + the vertical scroll distance.
MouseclickX	between 0 and [screen-width]	The X position of the mouse in relation to the monitor for a mouseClick event.
MouseclickY	between 0 and [screen-height]	The Y position of the mouse in relation to the monitor for a mouseClick event.
MouseclickPageX	between 0 and infinite	The X position of the mouse in relation to the page for a mouseClick event. This is effectively MouseclickX + the horizontal scroll distance.
MouseclickPageY	between 0 and infinite	The Y position of the mouse in relation to the page for a mouseClick event. This is effectively MouseclickY + the vertical scroll distance.
ClickedElement	e.g. ”fahrt_buchen”	The ID of an html element (for example, a button) that was clicked on.
URL	e.g. ”http://localhost:3003/app/#!/views/fahrt-buchen”	The address of the current page.

7.3 Design Specification

Last modified on 2017-11-08 (except for translations). Note that certain specifications such as the need for specific IIS settings in Windows are no longer applicable.

1. Current State
    - "User-Experience Optimization Prototype based on Emotion and Gaze Analysis in the Field of E-Commerce"
        - Contains code for retrieving Tobii Eye Tracker data (using an old SDK) as well as Noldus Facetracker data and sends this data to the web interface via polling
        - Reacts to certain situations using the Tobii EyeTracker and the Noldus FaceReader application
        - Contains one hard-coded "situation pattern" wherein an Angular.JS web application can change the contents of a website based on sensor data
2. Goal / Desired State
    - Collection of data from additional input device: an Empatica E4 Wristband
    - Expansion of the existing SitAdapt platform to a generalized Situation-Recognition and UI-Adaptation platform
    - Development of an example application in cooperation with L. Biawan Yameni
3. Product Environment
    - Features and parameters of the product
        - Flexibility in terms of hardware, as the connected hardware might vary (e.g. the FaceReader, Eye Tracker or Wristband might be missing)
    - End User
        - The main user of the product will be UI-researchers (such as, for example, J. Meisner). These will know how to use the software or receive schooling. Therefore, there are no special expectations regarding the visual appearance of the interface.
    - Operational environment
        - Operating System
            * Windows 7 Professional 64 Bit
        - Hardware of the target system
            * CPU: Intel Xeon E5-1620 @ 3,60 GHz
            * RAM: 16 GB
            * GPU: Nvidia Quadro 2000
        - Software of the target system
            * Windows
                · Specific Internet Information Services settings

        - Application always needs Administrator rights
      * Noldus FaceReader 7
      * Tobii EyeTrackerManager (for calibration)
      * Empatica BLE Server
    - Architecture of the System
      * Optional: Separated Server/Client PC for external observation (limited to one client)

4. Functional Requirements
   - Recording of data from additional hardware (Empatica E4 Wristband)
   - Consolidated logging of all data (the format can be arbitrary)
     - EyeTracker
     - FaceReader
     - Wristband
     - Metadata
   - The recognition of situations only takes place LIVE.
   - User interfaces (These are optional and are not a required part of the project)
     - Start window
       * Method to define situations and the respective UI-changes (for example, via an editor)
       * Method to enter other relevant settings (hardware configuration, IP addresses)
       * Status Display
         - ("Preview") of data
         - Detailed feedback about errors
       * Button to synchronize the initialization of data recording
     - Live-Observation
       * If a server-client architecture exists, show a live stream of the browser
       * Visualization of incoming data
     - Offline-Playback ("Playback Component")
       * Playback of user actions (E.g. per screen recording)
       * Video recording of the user
       * Visualization of the recorded, synchronized data
       * Playback of situation recognition
   - Triggering of user-defined adaptations in the application domain
     - A generalized interface to trigger changes in various domains is to be designed in cooperation with L. Biawan Yameni and C. Herdin

5. Non-Functional Requirements / Quality Requirements
    - Usability
        - Graphical user interface with visualization of data
    - Reliability
        - Reaction to severed connections
        - Handling (and monitoring) of corrupt data
    - Real-time tolerance
        - Maximum reaction latency: 100ms
    - Accuracy
        - Due to the partially unstable pulse signal (+/- 10 bpm), expanding [the system] to include a dedicated hardware component might be possible, but has low priority
    - Extensibility and adaptation
        - Planned extensions
            * Analysis of correlations between emotional states and user input (”big data analysis”)
            * Evaluation of the effectiveness of UI-Changes
        - Domain: For now, just for popular web frameworks (Angular, React, etc.)
6. Scope of Delivery
    - Deadlines
        - By 2017-11-15: Reliable recording of E4 Wristband data
        - By 2018-02-18: Submission of the application (at the same time as the bachelor’s thesis)
    - Format
        - Executable program
        - Instruction guide
        - zipped folder of the product including documented source code
        - potentially an image / saved state of the development environment

7.4 Installation Instructions (in German)

Eine Inbetriebnahme von SitAdapt 2.0 erfordert folgende Programme bzw. Laufzeitumgebungen:

- Microsoft Visual Studio Community 2017 (Link)
- Node.js (Link - bzw. auch auf der CD enthalten) - das Projekt wurde mit Version 6.11.3 erstellt.
- Ein Git-Client, z.B. GitHub Desktop oder Git-SCM, falls das Projekt über den GitLab Server der Hochschule bezogen wird
- DBMS RethinkDB Version 2.3.6 (Link - bzw. auch auf der CD enthalten)
- Google Chrome (getestet mit Version 64.0)

Um sich mit den diversen Hardware-Sensoren zu verbinden (bzw. um sie zu konfigurieren), sollten außerdem folgende Programme installiert werden (zusätzlich zu den notwendigen Treibern bzw. Installationsassistenten):

- Noldus FaceReader 7.0 mit folgenden Einstellungen
  - Unter File->Settings->Data Export
    - Aktivieren von Enable External Control
      - Port 9090
      - Aktivieren von ”export with a fixed interval”
    - Aktivieren von ALLEN unter dem Stichwort Export (Detailed log...) angegebenen Datensätzen
  - Unter File->Settings->Analysis Options ALLE Klassifizierungsoptionen unter ”Optional Classifications” aktivieren
- Tobii Eye Tracker Manager (Link - bzw. auch auf der CD enthalten) für die Kalibrierung des Eye Trackers
- Empatica BLE Server (Direkter Link - bzw. auch auf der CD enthalten) (Beschreibung) um eine Verbindung mit dem Wristband herzustellen

Der C#-basierte Teil der Applikation (im Ordner RecordingComponent) verwendet Abhängigkeiten, welche über den Visual Studio-internen Nuget Paketverwaltungsdienst zu beziehen sind: Im Projektmappen-Explorer mit Rechtsklick das Projekt auswählen, ”NuGet-Pakete verwalten” klicken, und dann im Suchfeld nach dem Namen des jeweiligen Pakets suchen. Die Version lässt sich mittels Dropdown-Menü aussuchen. Folgende Pakete müssen installiert werden:

- Tobii.Research.x86 1.1.4
- WebSocketSharp 1.0.2
- Newtonsoft.Json 9.0.0

Als zusätzliche Bibliothek wird die Datei FaceReaderAPI.dll benötigt, welche sich im myLib Ordner befindet.

Der JavaScript-basierte Teil der Applikation beruht auf eigenen Abhängigkeiten. In folgenden Ordnern muss jeweils einmal der Befehl ”npm install” ausgeführt werden um die benötigten Module vom Server des Node Packet Managers herunterzuladen. Warnhinweise können ignoriert werden:

- DatabaseWriter
- RuleEditor
- EvaluationAndAdaptation
- Kaffee (Dieser Ordner entspricht der Webapplikation von L. Biawan Yameni)

7.5 User Manual (in German)

7.5.1 Konfiguration der IP Addressen

SitAdapt 2.0 verfügt über zwei Bedienungsmodii. Entweder kann das System ausschließlich auf einem Rechner (”die lokale Maschine”) ausgeführt werden, oder das System kann sich auf zwei Maschinen verteilen. Im letzteren Fall würde die Frontend-Webapplikation auf einem zweiten Rechner durch einen lokalen Webserver bereitgestellt werden. Je nach Inbetriebnahme müssen zunächst IP-Adressen aneinander angepasst werden. Dies gilt nur für Komponenten, die nicht auf dem gleichen Rechner miteinander kommunizieren (Die RecordingComponent, der RuleEditor, der DatabaseWriter und die EvaluationAndAdaptation Komponente kommunizieren immer auf dem gleichen Rechner miteinander). Konkret: Auf dem Rechner, auf dem die Frontend-Webapplikation von Biawan Yameni zum Einsatz kommt, muss in Zeile 29 von Kaffee/newFolder/ mystatic/app/app.js die IP-Adresse auf die IP-Adresse des Rechners geändert werden, auf dem WebSocket Server (der DatabaseWriter) läuft.

Obwohl es nicht notwendig sein sollte, die Ports aneinander anzupassen, ist in Tabelle 5 eine Auflistung der relevanten Default-Einstellungen zu sehen.

Dienst	Port
Chateau / RethinkDB Admin Weboberfläche	3000
SitAdapt Rule Editor (Dev Build)	3000*
SitAdapt WebSocket Server	8088
Standard RethinkDB Admin Weboberfläche	8080
Noldus FaceReader API	9090
Empatica BLE Server	28000
RethinkDB Client Driver	28015
Local WebApp Server	8009

Table 5: Ports der relevanten Dienste

* Eine Besonderheit besteht: Port 3000 wird von dem SitAdapt Rule Editor nur verwendet, wenn der Port noch nicht durch ein anderes Programm belegt worden ist. Ansonsten wird eine Ausweichung auf Port 3001 angeboten, welche manuell bestätigt werden muss (”Y/N?”). Die Rethinkdb-Administrations-Alternative Chateau bietet keinerlei Ausweichung an und versucht immer Port 3000 zu belegen. Daher sollte Chateau immer vor dem RuleEditor instaziiert werden.

7.5.2 Konfiguration und Vorbereitung der Sensoren

Im Visual Studio Projekt SitAdapt2 muss die Datei Start.cs angepasst werden, um die verschiedenen Hardware-Sensoren aus- bzw. anzuschalten. Hierfür müssen die Zeilen 14 bis 16 entsprechend ausgeklammert werden. Die Bezeichnungen der verschiedenen Initialisierungsklassen entsprechen den jeweiligen Sensoren. Es kann eine beliebige Kombination an Komponenten hinzugefügt und entfernt werden.

Um die Ergebnisse von Noldus FaceReader als Eingabequellen zu verwenden, muss innerhalb von Noldus FaceReader (nach Installation und Anschluss des Hardwareschlüssels) eventuell zunächst ein neues Projekt (und ein neuer Teilnehmer) erstellt werden. Als nächstes wird eine Kamera-

Analyse hinzugefügt, dessen Einstellungen bestätigt werden müssen (Als Testwerte wurde mit der Logitech HD Pro Webcam C920 eine Auflösung von 1920x1080 und eine Frame Rate von 30,0 fps ausgewählt). Eine Analyse soll explizit NICHT gestartet werden (das übernimmt der Recording Component).

Um den Tobii EyeTracker als Eingabequelle zu verwenden muss (nach der Installation) die Tobii Recheneinheit eingeschalten sein. Als nächstes sollte im Tobii Pro Eye Tracker Manager der Eye Tracker kalibriert werden, auch wenn dies nicht zwingend erforderlich ist.

Um das Empatica E4 Wristband als Eingabequelle zu verwenden, muss der Empatica BLE Server gestartet (und der USB Bluetooth Adapter angeschlossen) werden. Anmeldedaten sind in 'empaticalogin.txt' zu finden. Als nächstes aktiviert man den Discover-Modus der Anwendung und drückt einmal auf den Knopf des Wristbands. Im Idealfall wird innerhalb weniger Sekunden eine Verbindung aufgebaut (in der Tabelle sollte dann ein Eintrag wie "a619f2" erscheinen). Bei Verbindungsschwierigkeiten wird empfohlen, die Empatica BLE Server Anwendung zu beenden und neu zu starten. Eventuell muss auch das Wristband heruntergefahren und neu gestartet werden, oder anders positioniert werden. Bezüglich des Informationsgehalts der LED der Armbanduhr wird auf [36] verwiesen.

7.5.3 Initialisierung der Programme

Um eine Session von SitAdapt 2.0 auszuführen, müssen folgende Aktionen durchgeführt werden (eine Installation gemäß Unterkapitel 7.4 wird vorausgesetzt):

1. rethinkdb.exe ausführen (je nach Download/Installationsverzeichnis). Ein Aufruf über die Eingabeaufforderung (im Folgenden Konsole genannt) wird empfohlen, um etwaige Warnungen und Fehlermeldungen wahrnehmen zu können.
2. Im Ordner DatabaseWriter mittels der Konsole "npm start" ausführen. Siehe 7.5.4 bezüglich möglicher Fehlermeldungen.
3. Im Ordner RuleEditor mittels der Konsole "npm start" ausführen (danach sollte ein Browserfenster starten mit einer URL wie "localhost:3000")
4. Falls der Einsatz von Hardware-Sensoren erwünscht wird, muss in Visual Studio das Projekt "SitAdapt2" im "Debug-Modus" gestartet werden. Auf den Rechnern der Usabilty Labors (64bit Betriebssystem) wird hierfür im Configuration Manager die Einstellung "x86" für die Active solution platform verwendet und "Any CPU" für die Platform des RecordingComponent Projekts.
5. Im Ordner EvaluationAndAdaptation mittels der Konsole "npm start" ausführen.
6. Im Browser den Tab des Regeleditors auswählen (meistens mit der URL "http://localhost:3000/").
7. Im Regeleditor ggf. Regeln anlegen, entfernen oder bearbeiten.
8. Im Regeleditor auf "Start Session" klicken. In der Konsolenausgabe des DatabaseWriters sollte jetzt eine Auskunft über die verschiedenen abonnierten Changefeeds zu sehen sein.
9. (je nach Aufbau) im Ordner Kaffee mittels der Konsole "npm start" ausführen.

10. (je nach Aufbau) im Browser den Tab der Webanwendung auswählen und lossurfen. Eventuell sollten die Ausgaben der Konsolenfenster der verschiedenen Applikationen im Auge behalten werden.

Wichtig: nach dem Beenden einer Session sollten außer RethinkDB alle Programme beendet werden (ansonsten bleiben Changefeeds aus vorherigen Durchläufen der Applikation weiter abonniert).

7.5.4 Bekannte Fehlermeldungen

Manchmal erscheint beim starten vom DatabaseWriter eine größere Fehlermeldung ”ReqlOpFailedError: Cannot perform read: primary replica for shard [””, +inf) not available in: r.table(”WristbandData”).delete()” o.ä, da die Datenbank noch nicht vollständig hochgefahren und einsatzbereit ist. Nach ein paar Sekunden sollte sich der Start-Befehl erneut ausführen lassen. Bei Verbindungsproblemen in Bezug auf den WebSocket Server wird ein Neustart aller Komponenten empfohlen. Bei schwerwiegenden Fehlern empfiehlt sich das Entfernen der Datenbank ’sitadapt’. Falls die Administrationsoberbläche von Rethink (siehe Konsolenausgabe von Rethink) nicht lädt, bzw. nicht zur Verfügung steht (wie es bei der Entwicklung von SitAdapt 2.0 auf den Rechnern des Usability Labors der Fall war), empfiehlt sich das Ausweichen auf 3rd-Party Werkzeuge wie Chateau.

Hinweise an Weiterentwickler Um den React / JSX-basierten Code des Regeleditors angenehm bearbeiten zu können, wird ein Editor mit der Fähigkeit, Code im ”Babel” Format farblich zu markieren empfohlen.