Developing Psychophysiologically Interactive Computer Systems

Toni Vanhala

University of Tampere
Department of Computer Sciences
Interactive Technology
M.Sc. thesis
May 2005

University of Tampere
Department of Computer Sciences
Interactive Technology
Toni Vanhala: Developing Psychophysiologically Interactive Computer Systems
M.Sc. thesis, 63 pages
May 2005

Abstract
This thesis presents a software framework to support the construction of psychophysiologically interactive computer systems. The framework was implemented in Java and C++ programming languages and it was used to construct two systems. The first system was a remote heart rate monitoring system. The second system was constructed for performing an empirical study involving both facial electromyographic and wireless electrocardiographic measurements. The first system was tested by four subjects who performed scenarios of voluntarily induced heart rate changes. Measurements from twenty-seven participants were collected with the second system. The results showed that the framework supported the construction of these systems and their accurate and reliable operation. The results also suggested that the framework supports extending these prototypes into robust real-world systems.

Keywords: biosignal processing, human-computer interaction, multimodal architectures, proactive computing, psychophysiology, software frameworks.

Contents

1. Introduction

The number of digital computers has been increasing since the late 1970s, when mass-produced computers were first introduced to the general public. Nowadays, computers are ubiquitous and many computers are embedded within our environment, clothes, and even our bodies [Tennenhouse, 2000]. Furthermore, the number of computers connected with each other and their surroundings is rapidly increasing. Consequently, the design, implementation, and evaluation of human-computer interaction are becoming more and more complex. One possible solution to this challenge is to introduce perceptive capabilities for the computers themselves [Pentland, 2000]. Being able to perceive their environment and classify the current situation, computers could support the goals of their human operators by anticipating future events and addressing them by taking appropriate actions. This would require that computers could also perceive humans, that is, detect the psychological and the physiological states of persons who are involved.

Although most of the previous research on human-computer interaction has focused on its technological and psychological aspects, psychology and physiology are interconnected and inseparable. Psychophysiology has long studied these connections that provide an opportunity to explore the mind through the functions of the body [Cacioppo et al., 2000]. For example, mental stress induces changes to heart functioning. The heart muscle generates electric signals that reflect these changes and propagate through the body [Brownley *et al.*, 2000]. The electric signals can then be measured from the surface of the skin using electrocardiographic (ECG) equipment. The intervals between successive heart cycles can be extracted from the acquired ECG data. Finally, the variability of these intervals can be used to evaluate the level of mental stress [Bernardi *et al*., 2000].

Psychophysiologically interactive computer systems perceive persons by collecting physiological data and extracting psychophysiological measures from this data. The systems use the extracted measures in order to select and provide appropriate feedback to the monitored person. The systems may also adapt their operation and functionality based on the acquired measures. Furthermore, the feedback that a system can give to a monitored person influences his or her physiology by affecting the psychological processes. The resulting changes of physiological functioning then consequently act as an input for the system [Figure 1].

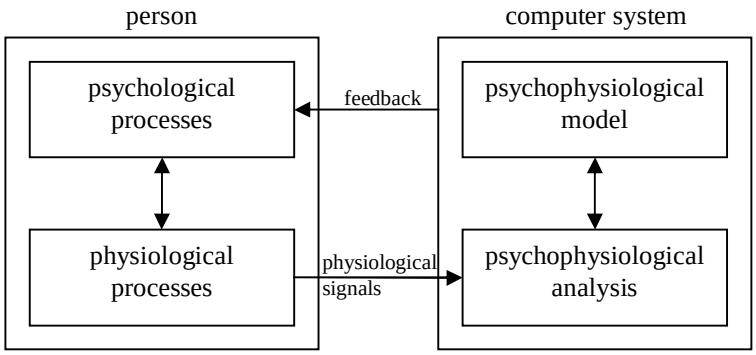

Figure 1. Psychophysiological human-computer interaction. The computer system analyses physiological data using a model of psychophysiological relationships.

The ability to continuously monitor psychophysiological processes differentiates psychophysiologically interactive systems from conventional computer systems, which view human-computer interaction as a processing loop [Tennenhouse, 2000]. Placed within the loop of human-computer interaction, humans must continuously and consciously communicate with a computer system in order to operate it. In psychophysiological human-coputer interaction, on the other hand, the monitored person does not need to actively participate in the interaction with the system as physiological signals are involuntarily and continuously produced. Thus, psychophysiologically interactive computer systems can support a person without distracting his or her tasks. A system provides this support by taking the initiative when required and appropriate, that is, by being proactive [Tennenhouse, 2000].

As an example, a system that monitors heart functioning might alert medical help in the case of a heart stroke. However, the person remains in the control of the system as he or she can influence the monitored heart signals. Generally speaking, humans can control their own physiology to a limited extent. Further, they can also be trained to better control their physiological processes. Heart functioning, for example, can be influenced with controlled breathing or by performing simple mental activities [Bernardi *et al.*, 2000]. Consequently, psychophysiological human-computer interaction is suited for the proactive computing paradigm, which views the human as a supervisor rather than an operator of computer systems [Tennenhouse, 2000].

Psychophysiologically interactive computer systems can support a wide range of applications due to their ability to utilize both voluntarily and involuntarily produced physiological data. Voluntarily controlled psychophysiological signals have been used, for example, to create methods for hands-free operation of computers [Surakka *et al*., 2004; Zhai, 2003; Millán,

2003]. As an example of the utilization of involuntarily produced psychophysiological data, Lisetti and LeRouge [2004] proposed physiological measures for identifying emotional states during medical data acquisition. There are many situations that involve diagnostic measurements and that can also influence emotions. One such situation is the common procedure of measuring blood pressure with a strap-on collar. Anxiety and stress induced by the situation can elevate the results. This elevation increases the risk of false diagnosis of a permanently elevated blood pressure. Thus, from a clinician's viewpoint, it is necessary to detect emotions in order to assess and eliminate their effect in diagnosis.

There is a wealth of different physiological signals. The most common psychophysiological measures are derived from bioelectric signals that are produced by nerve and muscle cells [Cohen, 2000]. Each of these signals has its own characteristics, for example, frequency range and magnitude [Table 1]. These characteristics require specific analysis methods to be used for each signal.

Table 1. Some common physiological signals with varying characteristics. Data compiled from Table 48.1 in Neuman [2000b] and Table 52.1 in Cohen [2000].

Physiological signal	Acquisition	Biologic source	Frequency range	Amplitude range
electrocardiogram (ECG)	surface electrodes	heart	0.05 – 1000 Hz	100 μV – 10 mV
electromyogram (EMG)	single-fiber EMG: needle electrode surface EMG: surface electrodes	muscle	0.01 – 10 kHz	1 μV – 2 mV
electroencephalogram (EEG)	surface electrodes	brain	0.5 – 100 Hz	2 – 200 μV
electro-oculogram (EOG)	surface electrodes	eye	0 – 100 Hz	10 μV – 5 mV
electroretinogram (ERG)	microelectrode	eye	0.2 – 200 Hz	0.5μV – 1 mV

For example, the smaller amplitude range of electroencephalographic (EEG) signals requires them to be amplified more than electrocardiographic signals, which have a much greater magnitude. Otherwise, the accuracy of acquired EEG would be greatly reduced. Further, physiological signals contain many different types of psychophysiological measures in many different analysis domains [Gratton, 2000]. Each of these domains is an independent source of information, although the domains also complement each other. Different signals and domains require different analysis methods. Thus,

psychophysiologically interactive computer systems are diverse in their requirements for signal processing.

In addition to the diversity and complexity of physiological signals, many other factors complicate the analysis of physiological data. These factors include the complexity of the human physiological systems themselves, indirectness of psychophysiological measures, and their context-dependency [Cohen, 2000; Cacioppo *et al.*, 2000]. One result from the complexity of physiological systems is that most psychophysiological processes (including, e.g., emotions) are reflected in more than one psychophysiological measure [Lisetti and Nasoz, 2002; Cacioppo *et al.*, 2000]. Furthermore, physiological responses to different psychological factors can be nearly identical [Ward and Marsden, 2003].

The indirectness of psychophysiological measures is the result of two characteristics. First, the tighter the coupling between the physiological process of interest and the sensor registering it, the more direct and noise-free is the acquired signal [Neuman, 2000a]. However, the tightness of the coupling is also related to the invasiveness of the measurement. Non-invasive measures are more practical, comfortable, and safe for the monitored person. Also, the sensors used in their acquisition are easier to maintain. For these reasons, psychophysiological signals are most often acquired non-invasively. Unfortunately, non-invasively acquired data has more noise than data that is acquired with invasive methods. This further complicates the extraction and analysis of psychophysiological measures.

Second, there is no clear, unambiguous linkage between mental processes and physiological activity. In comparison, the relationship between physiology and human health is extensively covered by models that can extract meaningful features of physiological functioning with relatively little effort. Actually, there is no generally accepted method for directly observing and measuring psychological variables, which operate inside the black box of human mind.

The context-dependency of physiological measures is evident in the variance of the base level of activity [Gratton, 2000]. The base level of activity can be defined as activity that occurs before the physiology responds to the psychological element of interest. The identification of the base activity level is difficult, even in controlled environments (e.g. empirical studies in a laboratory). When physiological data is to be used in psychophysiological human-computer interaction, this identification is even more difficult. In real-world applications there usually are no controlled epochs with certain identifiable conditions. However, these conditions do affect the base level of

activity [Cacioppo *et al.*, 2000]. Thus, context must always be taken into account in the analysis of psychophysiological data.

The context-dependency of psychophysiological data has recently become even more pronounced, as wearable, wireless, and mobile physiological monitoring devices have become common [Teller, 2004; Vehkaoja and Lekkala, 2004]. Mass-produced physiological monitors for the end-user are already available for several applications, including weight management and sleep monitoring [Bodymedia, 2005; Compumedics, 2005; Polar, 2004]. These new devices can operate in multiple contexts, which poses new challenges for psychophysiological computing. Psychophysiologically interactive systems that utilize these devices must repeatedly answer questions about who employs computation, where computation is performed, how people and devices interact, and what the computation is used for [Fitzmaurice *et al.*, 2003].

The challenges in analyzing psychophysiological data and utilizing it in human-computer interaction complicate the development of psychophysiologically interactive systems. The present thesis presents a software framework that aims to support the development of psychophysiologically interactive systems by addressing these challenges. The framework provides this support by offering a set of software components that different psychophysiologically interactive computer systems can share. Furthermore, the framework is implemented according to a set of design patterns that provide viable solutions for the software architectures of these systems.

The structure of this thesis follows the process of creating the framework. For designing the framework, the common requirements of psychophysiologically interactive computing systems were first identified. Then, a suitable architecture for handling many types of data processing and static system configurations was designed. In order to support the dynamic operation of systems, software agent technology was used to implement this architecture. Finally, two psychophysiologically interactive systems were constructed with the framework. The framework was evaluated based on the results from the implementation and operation of these systems.

2. Psychophysiological computing

2.1. Applications

The applications for psychophysiological data cover many diverse fields, including new interaction modalities for human-computer interaction, affective computing, and medical applications. Allanson and Fairclough [2004] divided these applications into biofeedback-based and cybernetically adaptive systems. Biofeedback-based systems provide feedback about physiological processes to the monitored person. The purpose of this feedback is to provide the person the ability to gain awareness of physiological processes. Being aware of these processes, the person is able to train for voluntary control over them. Adaptive biocybernetic systems, on the other hand, modify their own functionality and appearance based on the psychophysiological state of the monitored person.

However, interaction can cover other actors in addition to a single person and a single measurement system. As an example of this kind of *extended interaction*, computer-assisted diagnosis has been extensively studied for several years [Rangayyan, 2001]. The clinician is informed about the physiological and emotional state of the patient, who is being monitored with remote technology [Figure 2]. The interaction loop is closed by the feedback from the clinician to the monitored patient. In this case, feedback from a computer system is mediated through and moderated by another person. The system is neither biofeedback-based nor biocybernetically adaptive as interaction occurs between multiple actors: the patient, the clinician, and the computer system.

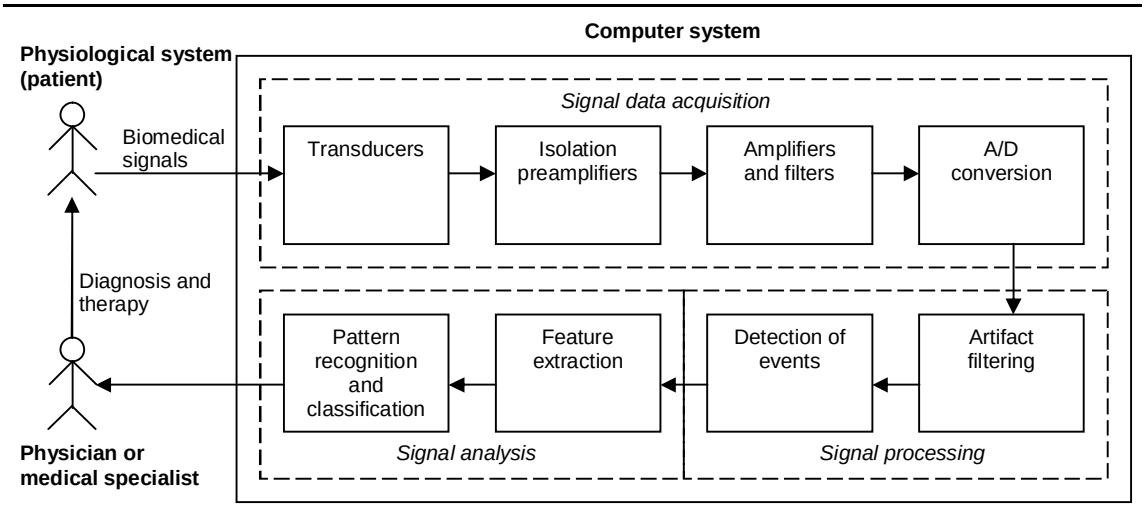

Figure 2. Computer-aided diagnosis and therapy based upon biomedical signal analysis [Rangayyan, 2001 (modified and redrawn from Figure 1.32)].

In addition to the type of interaction it supports, a system can be classified according to the type of data it processes. Systems can differ in both the amount and the abstraction level of data they require for their operation. For example, computer-aided diagnosis systems have to provide physicians with high-level data that aids their diagnostic decisions. Thus, as presented in Figure 2, a diagnosis system has to perform a lot of signal analysis steps prior to providing suggestions for the diagnosis. Table 2 presents a classification of psychophysiologically interactive systems according to the characteristics of data and the type of interaction they support.

Table 2. A classification for psychophysiologically interactive systems.

type of processed data		type of supported interaction: biofeedback-based	biocybernetic adaptation	extended interaction
low-level	short-term	clinical biofeedback	hands-free control using facial muscles and gaze	phobia treatment
low-level	long-term	fitness and weight management	virtual coaching	telemedicine
high-level	short-term	brain-computer interface	adaptive brain interface	attention monitoring
high-level	long-term	stress management	adaptive environments	social awareness

Biofeedback has been extensively used in clinical applications. In the first clinical application, feedback derived from electromyographic (EMG) activity was used to treat muscle tension headache. Tones and clicks corresponding to the tension of a forehead muscle were presented to the patient, who was trained to gain voluntary control of the muscle. Later, this approach was extended to general stress management. Currently, electromyographic biofeedback is a standard procedure in rehabilitation of nerve and muscle damages. During rehabilitation sessions, feedback derived from the EMG signal is used to guide the tension of spastic or denervated muscles [Tassinary and Cacioppo, 2000]. It is evident that the provided feedback must be

immediate and directly correspond to the intensity of physical activity. Thus, clinical biofeedback applications deal with low-level data collected during a short time period.

As an example of extended interaction based on low-level short-term physiological data, virtual reality environments combined with monitoring of psychophysiological activity were successfully used in treating a number of different phobias [VRMC, 2005; Garcia-Palacios *et al.*, 2002]. In this application, biofeedback was mediated by clinicians who monitored their patients during a treatment session. The anxiety and stress level of a patient were estimated from the registered physiological signals during the session. These estimates guided the clinician in adjusting the treatment for each individual patient and treatment session. Parameters of physiological activity were sometimes also shown to the monitored person. This enabled training the person to recognize and control the physiological responses associated with his or her fear.

Biocybernetically adaptive systems that process short-term data at a low level enable new interaction techniques. The motivation for developing such techniques is their potential efficiency, comfort, and accessibility for persons with disabilities. One approach is to combine information of where the attention is directed with voluntarily produced physiological activity. This approach was used in the work of Surakka and others [2004]. In this novel input method objects could be selected by looking at them and voluntarily activating a facial muscle. In other words, the system collected information about the visual attention (i.e., gaze direction) and the intent (i.e., voluntary facial action) of the user. This information was converted into interface commands. Thus, the method utilized the collected data in a very direct way, that is, without extracting any high-level meaning from it. The actions of the user were interpreted as events that changed the operation of the system, that is, adapted the system. As another example of a similar biocybernetically adaptive application, Felzer and Freisleben [2000] implemented a system for driving a wheel chair using voluntary activations of certain facial muscles.

Long-term monitoring of physiology enables a completely different set of applications. The information gained during a longer time period is not necessarily better nor more accurate, but different from what can be immediately deduced. For example, the frequency spectrum of electromyography provides information about the fatigue in the monitored muscles [Tassinary and Cacioppo, 2000]. Amplitude analysis, on the other hand, provides information about the current tension in the muscle. However, frequency analysis is based on data from a longer time period than amplitude

analysis. Thus, the type of information provided by the EMG depends, ultimately, on the length of the inspected time period.

An example of a system based on delayed biofeedback (i.e., feedback based on long-term physiological data) is the HealthWear weight loss system [BodyMedia, 2005]. The system monitors several physiological signals with wearable sensors. The activities and caloric expenditure of the person are derived from the registered signals. This data can then be used for balancing the intake and expenditure of calories.

Delayed biofeedback has also been used for fitness management. Polar [2004] provides a heart rate monitoring system that consists of an electrocardiographic (ECG) sensor that is worn around the chest and a wristwatch that collects and displays the ECG data. The system includes features both for measuring current level of fitness and for planning exercises accordingly. Both the HealthWare system and the fitness management system require long-term monitoring of physiology in order to estimate stable effects and trends in physiological processes. Short-term measures would be subject to many sources of measurement errors and artifacts and thus unsuitable for this purpose [Binkley, 2003].

As another example of a fitness-related application, Ijsselsteijn and others [2004] studied the effects of coaching on the motivation of participants who were cycling on a home exercise bike. The participants were given feedback on their performance every minute. The feedback was based on the heart rate of the participant and it was presented by a virtual embodied agent. In other words, the system adapted its operation (i.e., the feedback) according to long-term physiological data. The empirical results of Ijsselsteijn and others [2004] showed that the given feedback lowered the pressure and tension perceived by the participant during the workout. These results are supported by the findings of Partala and Surakka [2004], who studied the effect of emotionally-significant interventions in a human-computer interaction scenario. They found a positive effect of interventions on user performance.

Long-term physiological monitoring has clinical applications as well. Bondmass and others [1999] showed that long-term monitoring of physiological data significantly improved the quality of life after a heart failure. In their study, physiological measures were taken at home and electronically transmitted to a remote medical center. At the medical center, health care personnel were alarmed if too large deviations were present in the physiological data. Thus, described according to the suggested classification of psychophysiologically interactive computer systems, the system collected low-level long-term physiological data that was used in extended interaction

involving the patient, the health care personnel, and the system. The results of the study showed that readmissions, the length of stay in hospital, and hospital charges were significantly decreased as a result of home monitoring.

Systems that automatically extract high-level meanings from physiological data and utilize them in psychophysiological human-computer interaction require a more detailed model of psychophysiological processes than previously presented applications. Brain-computer interfaces, for example, are based on voluntarily produced brain activity [Hinterberger *et al*., 2004; Wolpaw *et al*., 2002]. Different mental tasks are performed by the user of the system. These tasks produce specific changes to the electrical activity of the brain, which enables them to be recognized by the system. Recognized tasks are converted into interface commands. The tasks can be used, for example, to move a cursor. Automatically recognized mental tasks could potentially be high-level data for psychophysiological human-computer interaction. However, in current brain-computer interfaces, the tasks that are used to operate the interface are fixed and limited. Furthermore, the user has to be trained to produce the required activity. Thus, classifying brain-computer interfaces as systems that process high-level data reflects more their future potential than their current capabilities.

Adaptive brain interfaces are similar to brain-computer interfaces, with the exception that they can adapt to mental tasks preferred by individual users [Millán, 2003]. However, this does not mean that adaptive brain interfaces perform a more detailed analysis of physiological data. The abstraction level of extracted measures is equal between brain-computer interfaces and adaptive brain interfaces. As a consequence, the users of an adaptive brain interface still have to be trained, but the training period can be considerably shorter compared to a conventional brain-computer interface.

Despite the current limitations in the automatic recognition of brain activity, Chen and Vertegaal [2004] have presented a practical application for involuntary, untrained brain activity. They used electric signals of the brain to distinguish between low and high states of motor activity. In addition to the motor state of the monitored person, the mental load of the person was also extracted from physiology. This estimate was based on the frequency characteristics of heart activity. More specifically, the total power in the lower frequencies of heart rate variability (LF HRV) was used to index the amount of mental activity. The measures derived from brain and heart activity were then combined in order to distinguish between four attentional states [Table 3].

Table 3. Classifying activities according to attentional state. (Redrawn and modified from Table 1 in Chen and Vertegaal [2004].)

	Low motor activity	**High motor activity**
Low LF HRV power	• Low mental activity • At rest *Candidate activities:* pausing, relaxation. *Mobile phone notifications:* ring. *Instant messaging status:* available.	• Low mental activity • Sustained movement *Candidate activities:* moving. *Mobile phone notifications:* ring. *Instant messaging status:* busy.
High LF HRV power	• High mental load • At rest *Candidate activities:* driving, reading, thinking. *Mobile phone notifications:* vibrate *Instant messaging status:* available.	• High mental load • Sustained movement *Candidate activities:* meeting, lecturing, writing. *Mobile phone notifications:* silent. *Instant messaging status:* busy.

The mobile phone of the person had different notification modes associated with each attentional state. For example, when the motor activity and the mental load were both low, received calls would cause the phone to ring. When there was little motor activity and a high mental load, the received calls would cause the phone to vibrate. In addition, each caller was notified of the person's status prior to calling. The notification was displayed with an instant messaging application provided for the caller on his or her desktop computer. This way, callers could themselves decide whether interrupting the person would be appropriate. Thus, the system supported extended interaction, in addition to being biocybernetically adaptive as well.

Psychophysiological measures also provide means for estimating the stress level of a person. An often used measure for mental stress has been the variability of the heart rate (see, e.g., [Hjortskov *et al.*, 2004] and [Chen and Vertegaal, 2004]). There is accumulating evidence suggesting that mental stress affects the onset and recovery from physical diseases, for example heart conditions [Strike and Steptoe, 2003]. In the long run, monitoring the mental stress level and providing feedback about it would enable the person to gain a better control over the factors that contribute to it. This would help in avoiding and relieving the effects of physiological diseases.

Reliable recognition of psychophysiological states can also be applied to infer individual preferences. For example, emotional reactions associated with particular songs could be used to recognize the likes and dislikes of a person. Similar information has been used to select background music that suits the diverse preferences of people who are, for example, working out in the same

gym [Chao *et al.*, 2004; Marti and Lee, 2000]. Thus, unobtrusively acquired psychophysiological measures of preferences could provide methods for automatically adapting shared environments.

This approach could be expanded even further by providing information about the patterns of psychophysiological states to other persons. First, the attentional state of a person would be inferred (e.g., with the method of Chen and Vertegaal [2004]). During long-term monitoring, these states form temporal patterns that can be extracted from the data [Fisher and Dourish, 2004]. It might be, for example, that a certain person is usually occupied with mentally and physically challenging tasks during mornings. Then, he could be more easily reached during the rest of the day. Thus, based on the patterns of attentional states, it is possible to infer and predict when a person is available for interruptions and when not. Of course, temporal patterns could be extracted also from other psychophysiological measures (i.e., besides attention) and these patterns then applied in a similar manner.

A number of different applications for psychophysiological data were presented in this section. The purpose of this discussion was to illustrate the potential of psychophysiological human-computer interaction and the diversity of its applications. As the previous examples showed, applications have differing requirements concerning the abstraction level and the amount of data they process and the types of interaction they support. Thus, the design and implementation of tools that support the construction of psychophysiologically interactive computer systems is a challenging task. In the following section, these challenges will be discussed in more detail.

2.2. Challenges and solutions

The challenges that the development of psychophysiologically interactive computer systems faces are related to both the nature of psychophysiological data and the broad range of potential applications. Thus, a framework that supports the development of these systems should incorporate basic tools for physiological signal analysis as well as support for many types of systems and their diverse software architectures.

The challenges that relate to the nature of psychophysiological data can be summarized as follows (the challenges are numbered for later reference):

1. Psychophysiological data is context-dependent. Information about the context is required to interpret the data. [Cacioppo *et al.*, 2000; Gratton, 2000]
2. The parameters of data acquisition have a large significance for later signal analysis [Mainardi *et al.*, 2000; Tassinary and Cacioppo, 2000]. As

an example, different standards for electrode placement have been defined for electromyographic and electroencephalographic measurements in order to guarantee their validity and comparability [Fridlund and Cacioppo, 1986; Böcker *et al.*, 1994].

3. Psychophysiological data is non-specific. Every physiological process is affected by a number of psychological factors and vice versa [Cacioppo *et al.*, 2000]. Furthermore, physiological responses to different factors can be nearly identical [Ward and Marsden, 2003].
4. Psychophysiological responses are individual. Thus, information about the individual is required to interpret psychophysiological data. [Allanson and Fairclough, 2004; Ward and Marsden, 2003]
5. Physiological data is noisy and recognition of psychologically significant events is unreliable. Often, several signals are collected and analysed in order to increase the validity of measurements. [Cohen, 2000; Oviatt and Cohen, 2000; Teller, 2004]
6. Different dimensions of psychophysiological data provide different types of information [Gratton, 2000]. Processing data in time, frequency, amplitude, and spatial domains must be supported.

The challenges related to the diversity of psychophysiologically interactive computer systems include:

7. Proactive and ubiquitous computing favours systems that are distributed, mobile, and embedded [Tennenhouse, 2000; Weisner, 1993]. In order to support these increasingly popular computing paradigms, it is necessary to provide support for the construction of systems that consist of diverse components [Allanson and Fairclough, 2004; Davies and Gellersen, 2002].
8. Due to the systems being distributed and mobile, the context in which a system operates may change unexpectedly. Systems must be context-aware and ready to adapt to different contexts. [Davies and Gellersen, 2002]
9. Many psychophysiologically interactive applications require constant, long-term monitoring of physiological signals (see, e.g., those discussed in the previous section). Related to the eighth challenge, systems must also be sensitive to changes in the physical environment and the software context in order to guarantee reliable monitoring. This requires that systems adapt themselves when the goals of and the tasks for the system are at risk.

10. Physiological data is processed at and presented with different levels of abstraction, partly depending on the role that humans have in the operation of a system. For example, clinicians prefer to have a higher abstraction level of data when they make diagnoses and prognoses [Rangayyan, 2001]. For this reason, multiple processing steps have to be completed before the data is presented to the user of a computer-assisted diagnosis system (see, e.g., Figure 2 in Section 2.1). On the other hand, clinical biofeedback is performed using low-level data that is immediately displayed to the patient [Tassinary and Cacioppo, 2000]. Thus, when dealing with psychophysiological data, different abstraction levels must be supported [Allanson and Fairclough, 2004].

There are two general methods that address these ten challenges. The first method is to combine several parallel input signals in order to achieve better validity and accuracy in analysis of data. The other method is based on the use of context as an additional source of information.

The combination of different signals, that is, modality fusion, can be approached in two different ways. The first approach is to combine the different sources of information at the feature-level. This type of fusion is performed at an early stage of analysis by combining signals that are temporally close to each other. The second approach is to combine independently recognized events at the semantic level. This approach is sometimes also called decision-level fusion. [Oviatt and Cohen, 2000]

In semantic fusion, the significant events are first recognized from each input stream. Then, the events are combined using semantic rules. Semantic rules are relatively easy to understand and define, when compared to those used in performing feature-level fusion. Rules that deal with feature-level data are usually extracted automatically with machine-learning and other artificial intelligence and data mining methods. As a consequence, they usually are not in a form that can readily be interpreted by a human supervisor. Thus, semantic fusion is preferred when human understanding of and control over the recognition process is desired. According to Oviatt and Cohen [2000], semantic fusion also requires less training data for a system.

As an example of modality fusion in the recognition of psychological states, Zeng and others [2004] studied the benefits of fusion in the recognition of emotions. They found that fusing prosodic cues of the speech and facial expression data improved the accuracy of emotion recognition, when compared to methods that used only one of these two modalities. The accuracy of recognition was 56 percent when a facial expression classifier was utilized

alone. Prosody-only recognition resulted in an accuracy of only 45 percent. When these two modalities were fused together, an accuracy of nearly 90 percent was achieved in the recognition of emotional expressions. The most accurate classifier fused the two modalities at the decision (i.e., semantic) level.

Similarly, Busso and others [2004] reported their experiment on recognizing emotions from facial expressions and speech. They inspected the accuracy of emotion recognition both when one of the two modalities was utilized alone and when the modalities were fused together. Fusion at the feature-level and at the semantic-level was studied separately. Figure 3 illustrates the results from this experiment.

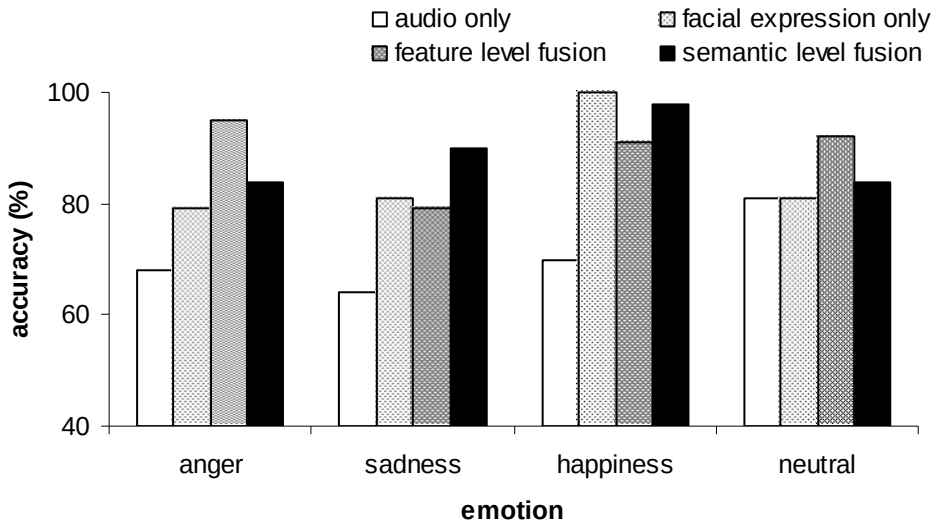

Figure 3. Recognition accuracies for four emotions using unimodal and bimodal classifiers. Accuracies were derived from the results of Busso and others [2004]. Accuracies shown for semantic fusion are those achieved using the semantic rule with the best overall performance.

To summarize, the results of Busso and others [2004] showed that both fusion methods improved the recognition of some emotions, but degraded the recognition of happiness compared to analysis of facial expressions alone. Feature-level fusion performed better for expressions of anger and neutral expressions. However, semantic-level fusion was more accurate in recognizing the other two emotions. Thus, it was not possible to determine, which approach to fusion was generally the better one.

The best suited fusion method might depend on the context. Thus, knowing the context could help in selecting the most appropriate method for analysis.

The context includes people, places and objects that are relevant for the current task [Dey, 2001]. Dealing with psychophysiological data, we should also include information about other situational factors that might affect the psychophysiological processes, for instance, the time of day [Allanson and Fairclough, 2004; Ward and Marsden, 2003].

Context is an important factor in the analysis of psychophysiological data in general. Cacioppo and others [2000] summarized the significance of context for psychophysiological analysis as follows:

> "… [A] wide range of complex relationships between psychological and physiological phenomena might be specifiable in simpler, more interpretable forms within specific assessment contexts."

As an example of how context simplifies psychophysiological phenomena, the individuality of physiological reactions can be taken into account by adapting to the context. The context provides cues that help in the interpretation of a psychophysiological reaction, even if the reaction is highly individual in nature. For example, a sudden change in parameters of heart activity might be due to many factors, including a heart failure and an emotional reaction. If the change in heart activity coincides with the person viewing a World Wide Web page with emotionally-significant content, there can be a greater confidence for the chosen interpretation, when compared to a situation in which no contextual clues are provided.

Further, collection and analysis of long-term data helps in estimating and anticipating the effects that a particular context has on an individual person. This way, an individual model can be formed for each monitored person. This enables systems to account for individual differences in psychophysiological reactions. In many cases this is essential, as general models of psychophysiology are not sufficient for the recognition of psychophysiological events that occur rarely and last for a short period of time [Ward and Marsden, 2003].

Finally, information about the context is crucial for systems that are distributed, mobile, or ubiquitous [Davies and Gellersen, 2002]. These systems have to adapt in response to variations in the availability of resources. For example, a chronically-ill person working in an office could be continuously monitored using wireless or ubiquitous technology. Her heart rate would be registered with a wireless electrode system (e.g., the system presented by Vehkaoja and Lekkala [2004]) that transmits the electrocardiographic data to a desktop computer. However, if she left her office and moved outside of the range of the wireless connection with the desktop computer, some other device would have to assume some of the tasks performed by the computer. This

could be a Personal Digital Assistant (PDA) that the person carries with her, for instance.

However, a PDA would not have as great signal processing capabilities nor network bandwidth as a desktop computer. Therefore the two devices would not be interchangeable in the system architecture, which would have to be modified. The PDA could for example be assigned with storing the data, until the wireless connection can again be established. Then, after re-establishing the connection, the PDA could deliver the stored data to the desktop computer. In order to appropriately adapt the architecture, the system would have to know the resources and services offered by both devices (i.e., the desktop computer and the PDA). In other words, the hardware and software context should be known to the system.

In summary, information about the context can help both to extract meaning from psychophysiological data and to automate the management of the system architecture. The former task can also be supported by providing multiple signals for psychophysiological analysis and utilizing methods that gain leverage from the complementing information in these signals. The next section discusses some of the previously applied software tools for this purpose, as well as tools for the development of psychophysiologically interactive computer systems in general.

2.3. Software

As early as in 1982, Arroyo and Childers presented their modular software system for the recognition of brain activity. The task of the system was to collect and classify single visual evoked potentials from electroencephalographic signals [Arroyo and Childers, 1982]. In order to support modularity, the system was constructed of several software programs. Each of the programs transformed the data to a form that could be further processed by another program. In other words, each of the programs solved a subproblem. The appropriate sequence of programs could then perform the overall task of the system. One of the design criteria of Arroyo and Childers was the generality of the system, that is, the ability to adapt and apply its parts to many systems and applications. The modularity of the system fulfilled this requirement. As the tasks and problems of the systems were decomposed to smaller parts, programs that solved the emerging subproblems could be used in many applications.

Currently, a large collection of software tools (i.e., a toolkit) is available from the Massachusetts Institute of Technology under the GNU General Public License (GPL) [PhysioNet, 2003]. This collection is called the PhysioToolkit and it includes tools for event recognition, acquisition of data, data visualization,

data conversion, and many other tasks associated with the utilization of physiological signals. These tools can be used much in the same manner as the software modules of Arroyo and Childers [1982]. As the PhysioToolkit is released under the GNU GPL, the source code of the tools is also open. This openness enables the toolkit's users to modify the tools in order to integrate them to their own systems. However, the license of the tools requires that the modifications and the resulting system have to be released under the GNU GPL as well. This might restrict their applicability to non-commercial use only.

A software architecture gives a high level description of the structure and operation of a software system [Schmidt *et al.*, 1996]. When modular tools are used as a basis, the system architecture complies with the Pipes and Filters design pattern [Buschmann *et al.*, 1996]. Systems that use this pattern consist of a sequence of programs that transform data. The result from a transformation is processed by another program that is next in the series.

The PhysioToolkit itself does not provide any method for constructing the architecture (i.e., defining the order of programs), nor the means to receive and send data between programs. However, the environment in which the tools are used may provide a method for defining the system architecture, that is, for joining the tools together. For example, the UNIX shell (i.e., text-based user interface) provides this functionality with a special pipe character ("|"). Sequential commands separated with the character are joined together. The resulting sequence is called a pipeline.

Output from a preceding command in a pipeline is provided to a succeeding command through the standard shell interface. For example, the combined result of the commands in Figure 4 is that the system sends a mail to the address "Some.One@Somewhere.biz". The mail contains the number of lines in the file "test.txt". The first program (*cat*) simply reads the file and sends it to another program (*wc*) that counts the lines. Finally, the line count is sent to the last program (*mail*) in the pipeline. It sends the received message (i.e., the line count) to the recipient via electronic mail.

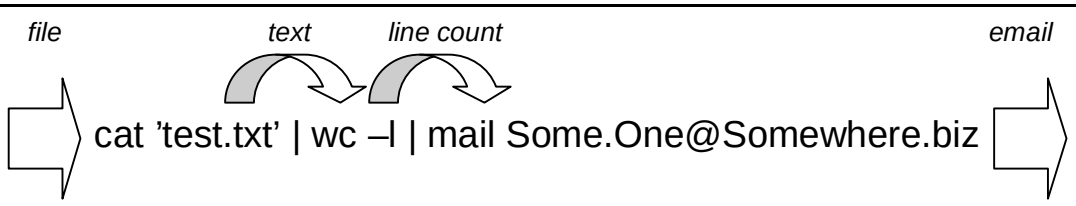

Figure 4. A UNIX pipeline.

Even if the environment does provide a method for defining the system architecture, the architecture must be defined and the environment known before the system is running. In other words, the PhysioToolkit does not provide means for real-time adaptation, nor does any other toolkit per se. Thus, the tool-based approach is not sufficient for biocybernetically adaptive systems and suits even worse for systems that have multiple purposes, that is, can adjust themselves to serve the (unexpected) needs arising from sources external to the system. As discussed in the previous chapter, this concerns most mobile, distributed, ubiquitous, and wearable systems. For example, wearable sensors that provide data for nearby systems would be difficult to include in architectures based on separate tools. Wearable sensors travel from a location to another with the person who wears them. As a consequence, the availability of external resources, such as wireless network connections and other devices, varies during the operation of the system.

As another example of existing tools for psychophysiological computing, Allanson [2002] presented a JavaBean toolkit for the development of physiologically interactive computer systems. JavaBean components enable the development and configuration of systems using a visual editor, such as the Bean Builder [Sun, 2004; CollabNet, 2004]. Visual editing may be especially suited for prototyping and less technology-oriented persons. Research systems that collect physiological data are often managed and configured by researchers specialized in psychology and physiology instead of programming. Thus, JavaBean components could be a feasible solution to support the construction of psychophysiologically interactive computing systems.

Although visual editing has its benefits, it can generally be used only to define the beginning state of the system. Real-time adaptation of systems that are constructed of separate components is restricted, regardless of the tools that are applied. On the other hand, defining multiple states for a system and transitions between these states is quite simple with a graphical editor. This would also enable the system to adapt. However, this is not practical for systems that are even moderately complicated. The reason for this is the large number of possible states and transitions that quickly add up to an unmanageable number of different combinations.

In addition to searching for specific tools for the utilization of physiological signals, it is possible to inspect existing systems and find architectural solutions and design patterns that are suitable for psychophysiological computing. Furthermore, software frameworks that have been used in the construction of these systems might provide leverage for the development of psychophysiologically interactive computer systems as well.

As discussed in Section 2.2, multiple physiological signals can be used to support psychophysiological analysis. This suggests that it would be appropriate to primarily focus on multimodal systems, as these systems are designed especially for this purpose. In addition to utilizing multiple parallel input signals, multimodal systems model the content of interaction at a high level of abstraction [Nigay and Coutaz, 1993]. As the psychophysiologically interactive systems must form psychological interpretations from physiological data, this is a necessity for them also. Besides the fusion of modalities and extraction of high-level data, there are also other relevant fields of research that the work on multimodal interaction has already covered. These fields include distributed systems, mobile systems, and adaptive systems. Thus, a closer inspection of multimodal systems could give an insight to the possible solutions for a number of challenges that multimodal and psychophysiologically interactive computer systems have in common.

A popular approach in the development of multimodal systems is to solve problems by employing a number of independent software agents. Although there have been many attempts to define an agent, none of them is generally accepted yet. According to Russell and Norvig [1995], an agent is an autonomous entity that perceives its environment through sensors and acts upon that environment through effectors. The behavior of an agent is determined by both its built-in knowledge and the experience it gains. In other words, agents have an internal state, which they update based on the actions they take and changes they perceive. This internal state enables agents to aim for a goal, anticipate future events, and take the initiative. Thus, according to the definition of Russell and Norvig [1995], all agents are proactive [Tennenhouse, 2000].

The QuickSet system is an example of an agent-based multimodal system [Cohen *et al.*, 1997]. The QuickSet system was developed for multimodal interaction using voice and gestures. It was implemented based on the Open Agent Architecture [Moran *et al.*, 1998]. This architecture supports multiple agents that can be written in many programming languages and run on different platforms. Each system contains a facilitator agent that handles requests from other agents, divides these requests into tasks, and delegates these tasks to agents that can perform them. A high-level language called Interagent Communication Language (ICL) is used for this purpose. The architecture also supports multiple facilitators. However, according to Moran and others [1998], multiple facilitators are seldom required.

The strong sides of the QuickSet architecture are its distributability and the support for multiple software and hardware platforms. Cross-platform

communication between agents is made possible by the high-level language the agents use to communicate with the facilitator and each other. On the other hand, the facilitator (or multiple facilitators) can form a bottleneck in systems where data is frequently interchanged [Moran *et al.*, 1998]. Thus, physiological data, which is collected at a high sampling rate, cannot be mediated through the facilitator.

As another example of agent-based architectures, Elting and others [2003] presented the Embassi system that was applied to multimodal interaction with consumer electronics, such as television receivers and home stereo appliances. The Embassi system used a layered grouping of agents. Layers processed information at different levels of abstraction. The modalities were independently analysed and fused together at the semantic level. Instead of using a central data structure or a facilitator agent for handling communication between agents, agents were organized to a pipeline, that is, information flowed from lower to higher abstraction levels. Information that concerned the whole system was provided by a separate context-manager.

Agents could join and leave the Embassi system at any point of its operation by informing the Polymodal Input Module, which was the component that performed the fusion of different modalities. This very straightforward approach was suitable for a system aimed for multimodal voluntary control of applications and hardware. The modalities complemented each other and when an agent left the system, input from the corresponding modality could simply be excluded.

However, this is not sufficient for every psychophysiologically interactive application. To recapitulate an earlier example, a person could wear a wireless electrocardiographic (ECG) sensor that measures her heart activity. Then, if she moved outside the range of the receiver, an agent reading the sensor would notice that the measurement is no longer valid and decide to leave the system. If the purpose was to register the heart rate and use it in the analysis of mental effort (e.g., based on heart rate variability), an intelligent system would not cease the measurement of the mental effort completely, but possibly store the ECG data for later analysis, or use another signal to evaluate the mental effort.

Furthermore, in the Embassi system, agents that analyzed other modalities were queried in order to perform the fusion of modalities, whenever input was received from one modality. This is not a generally suited solution for psychophysiological human-computer interaction, as it forces the systems to use semantic-level fusion and the recognition of significant events is difficult from any single physiological signal or other modality (see, e.g., [Cacioppo *et al.*, 2000; Ward and Marsden, 2003]).

This section presented software architectures that have been used to address challenges faced by psychophysiologically interactive computer systems. Although an answer to every challenge in psychophysiological human-computer interaction was not found, the presented architectures suggested solutions that can be useful when developing psychophysiologically interactive computer systems. Table 4 summarizes the challenges of psychophysiological computing and solutions offered by the existing tools.

Table 4. Challenges for psychophysiological computing and solutions offered by existing architectures (numbering corresponds to Section 2.2.).

Challenge	**Toolkits (pipelines)**	**Agent-based architectures**
1. Psychophysiological data is context-dependent.	- No method provided for acquiring and analyzing context.	+ A separate agent may be provided for managing context.
2. Parameters of data acquisition must be known in analysis.	- No support offered for defining parameters and preserving them through processing.	+ Flexible inter-agent language enables the agents to communicate parameters at a high level.
3. Psychophysiological data is non-specific.	- No method for dealing with ambiguity. Focus is on the analysis of a single signal.	+ The fusion of parallel signal helps to resolve ambiguities. - The provided method for signal fusion is inefficient for the processing of low-level data (see also challenge #10).
4. Psychophysiological responses vary between individuals.	- No support offered for storing and taking into account individual parameters.	+ The agent that manages the context can provide information about the individual. + Individual parameters may be preserved or queried through processing.
5. Recognition of events is unreliable.	See the third challenge.	+ Context-awareness and signal fusion help to resolve ambiguities.
6. Different domains of data and analysis must be supported.	+ Components can be replaced in order to analyze different domains. - Simultaneous analysis of multiple domains is not supported.	+ The same data can effortlessly be provided for multiple agents that analyze different domains at the same time.
7. Systems are often distributed.	- Toolkits themselves do not provide methods for distributed computing.	+ The communication between agents is independent of software and hardware environments.
8. Systems must be context-aware and adaptable.	- No support for context-awareness. - Only static architectures are supported.	+ Modifying the architecture is possible. + The most suitable agents are recruited for performing a task at a particular time.
9. Support for long-term monitoring must be included.	- The constructed systems do not have awareness of the properties and status of individual components (i.e., tools).	+ Changes in the context can be taken into account. - The adaptability of system architectures is limited.
10. Different abstraction levels for processing and communicating physiological data should be supported.	+ The type and level of data passed between components is not fixed. - No method provided for coding the abstraction level of data.	- The central agent that manages the architecture (e.g., in QuickSet) or performs signal fusion (e.g., in Embassi) forms a bottle-neck for low-level data.

It should be noted that only solutions offered by the approach in general are presented in Table 4. For example, although an individual tool might provide a method for analyzing context, using a toolkit does not guarantee that ability for every system constructed with it. As Table 4 shows, psychophysiological human-computer interaction has some specific requirements that these architectures do not address. These needs are addressed in this thesis by constructing a framework that is specifically intended for the development of psychophysiologically interactive computer systems.

The design of a software framework begins with the identification of functionality that is common for applications in the domain of interest, in this case, psychophysiological interaction with a computer system [Flippo et *al.*, 2003]. This was done both by inspecting the previously discussed applications (Section 2.1) and by analyzing some existing software tools in this section. Next, a core that does not contain any application-specific functionality is to be defined. Finally, the framework is to be implemented and evaluated. The remaining steps are taken in the third and the fourth chapter.

3. Methods

3.1. Framework

3.1.1. Structure of the framework

As discussed in the previous section and illustrated in Table 4, no readily available solutions exist for all of the challenges of psychophysiological computing. However, several partial solutions do exist. Thus, instead of utilizing a single method, several approaches have to be combined in order to create a framework that adequately supports psychophysiological human-computer interaction. In the present work, the focus was first on designing a method that would enable the construction of stable architectures from modular components. Then, this method was extended with the ability to adapt architectures during their operation.

Pipelines are suited for processing of physiological data due to their efficiency and support for the reuse of components [Buschmann *et al.*, 1996; Ilmonen and Kontkanen, 2003]. For this reason, the Pipes and Filters design pattern was selected as the basis for composing static architectures with the framework. In this design pattern, data flows through pipes that run between filters. The pipe is an abstract concept for the connection between filters and does not force any particular implementation to be used. Filters transform the data they receive, process the data, and send the result through an outgoing pipe. Thus, a system consists of pipelines [Figure 5].

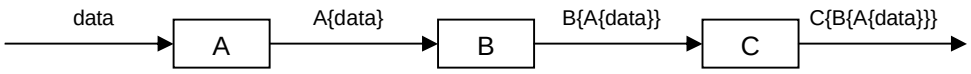

Figure 5. Information pipeline. The data is fed to the system through the first filter, which performs some transformation on the data. The resulting data is then fed to the second filter. Finally, the result from these two transformations is fed through the third filter. The output of the system is the combined result of these three transformations. If filters are viewed as mathematical functions A{x}, B{x}, and C{x}, the system corresponds to the composite function C{B{A{x}}}.

In order to support psychophysiologically interactive systems that can consist of more complex pipelines, the basic Pipes and Filters pattern was extended in the present framework. This extension enabled systems to handle architectures that support sending information to preceding filters, as well as architectures that allow the processing flow to be split into separate flows or several flows to be joined into a single one [Figure 6]. The benefits of these more complex architectures include increased efficiency due to the possibility to share filters between processing flows. Another benefit is the adaptability that results from the ability to provide feedback to earlier stages of processing.

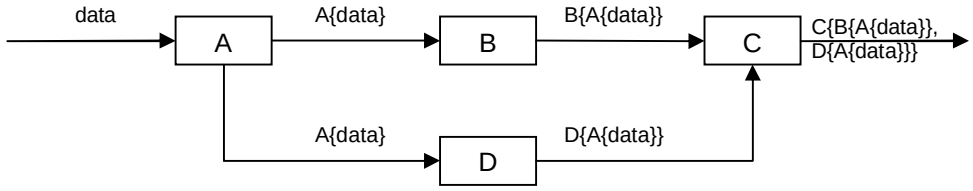

Figure 6. A complex processing flow. The flow is split at the first filter and rejoined at the third filter. Data could be fed back to preceding filters, but these types of connections are left out for clarity of presentation.

The connections (i.e., pipes) between filters were available through buffers. Each filter contained a separate buffer for each of its input and output channels [Figure 7].

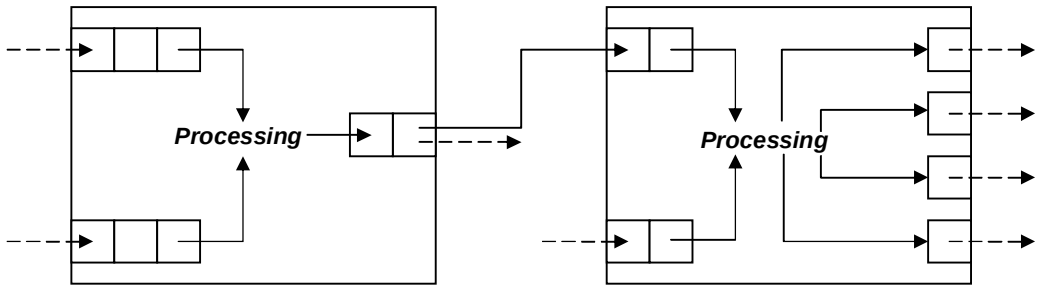

Figure 7. Two filters connected with a pipe. The filter on the left has two input channels and one output channel. It provides data for both the filter on the right and another filter that is not displayed in this figure. The filter on the right receives data from the left filter and from another filter, not displayed here. The filter produces four different outputs from the two inputs.

Processing items could be retrieved from incoming buffers, processed, and the results placed in an outgoing buffer. The framework handled the actual delivering of items from a filter to another. However, each filter was responsible for flushing its outbound buffers when they were full.

Managing the connections between filters can be very complex, especially when the filters can dynamically change their processing and the architecture by modifying themselves and joining or leaving the system during its operation. Changes to one part of the system can affect its other parts, which impedes the search for the optimal software architecture. For these reasons, in addition to the pipes and filters, a centralized and more abstract method is required for managing the architecture dynamically (i.e., while the system is operational). To address this need, each filter was encapsulated in an agent that managed the respective filter. This way, the framework could take advantage of both the efficiency of the static pipeline-based architecture and the adaptability offered by software agents.

Every agent registered to a central agent called the Broker. During the registration, an agent described its processing capabilities as well as the properties of its input and output channels. The communication between filters and the Broker was handled using a high-level language based on Extensible Markup Language (XML) [W3C, 2005]. Figure 8 presents an example of a typical registration message.

```
<?xml version='1.0' encoding='utf-8'?>
<register>
        <IP>
                127.0.0.1:50004
        </IP>
        <id>
                CORRELATOR
        </id>
        <input>
                <id>
                        ECG
                </id>
        </input>
        <output>
                <id>
                        HEART_RATE
                </id>
        </output>
</register>
```

Figure 8. An example of a registration message in the XML-based language.

The Broker managed the connections between filters following the Mediator design pattern [Gamma *et al.*, 1994]. When a new pipe was formed between two filters, the Broker asked the agent that managed the receiving filter to prepare for the incoming data. Then, the Broker provided the sender the necessary information about the hardware and software environment of the receiver. The sender formed a connection to the receiver and informed the Broker of the result, that is, whether the connection attempt to the receiver was successful or not. Removing a pipe from the architecture was performed in the opposite order (i.e., by first informing the sender and then the receiver of the data).

3.1.2. Implementation of the framework

The framework was implemented in Java and C++ programming languages. The implementation consists of an abstract base class for agents, classes that extend this base class for different types of filters, the Broker, and agents that implement specific data processing methods for psychophysiological signals. The base class for an agent is available in both languages. The extensions of this class are implemented in C++ and contain functionality for agents that send data, receive data, or convert data and pass it forward (i.e., both receive and send data). These classes are called the Sender, the Receiver, and the Filter class, respectively. Class diagram of the C++ implementation is shown in Figure 9.

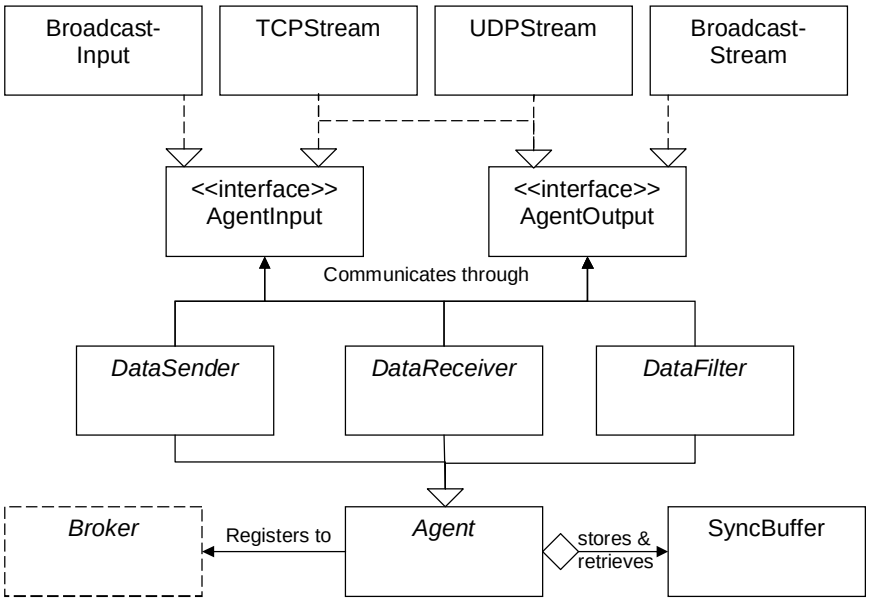

Figure 9. Class diagram of the basic components in the framework. The presented classes were implemented in the C++ language, except the Broker, which currently has only a Java implementation.

The connections (i.e., the pipes) between filters are accessed through abstract interfaces that hide the true implementations of the data stream. Data is received via the AgentInput interface and sent via the AgentOutput interface. The underlying pipe is implemented either as a User Datagram Package (UDP) or as a Transport Control Protocol (TCP) stream. Connections used only to either receive or transmit data are also supported. Currently, support for sending and receiving UDP broadcasts is implemented as BroadcastStream and UDPInputStream classes, respectively.

Implementing other types of connections (in addition to UDP and TCP streams) would simply require the developer to extend one or both of the interfaces for receiving (AgentInput) and sending (AgentOutput) items. All communication between agents is handled through these abstract interfaces. This way, the underlying implementation of a stream can be changed without modifying the algorithms that handle the communication via the stream. Furthermore, the current implementation of components, including streams, uses a multi-platform program library to access basic services of the operating system and another for managing the components that provide a graphical representation [Sugar, 2005; Smart, 2004]. These libraries enable the same components to be compiled and run under multiple environments without modifying the components themselves.

A number of filters were implemented for a number of specific psychophysiological signal processing tasks, including correlation calculations and the computation of the power of a signal. However, there are a large number of methods and algorithms that could be implemented and many applications have specific requirements. Thus, it is impossible to anticipate and address every need. As a result, the developer has to occasionally create new components components in order to cover the requirements of the application at hand.

A new filter can be created by extending either the base Agent class or one of the more specialized classes (i.e., the Sender, the Receiver, or the Filter class). In practice, extending one of these classes consists of overriding the XML presentation of the agent and implementing the data processing that is specific for each filter. The filter can read data from an internal buffer and place processed data into another buffer. These buffers are provided by the framework. Furthermore, if one of the specialized classes is used as the basis for the extension, the developer is released of the effort of retrieving new processing items from the data stream (i.e., pipe), placing these items to the incoming buffer, and sending the data placed in the outgoing buffer to recipients. The framework handles each of these steps.

Figure 10 presents a typical lifespan of an agent as a sequence diagram [OMG, 2005]. Lifespan in this case means the period of time when the filter is an active part of the system, that is, connected to it in some way. The actual software component that corresponds to the agent might be active and reside in memory outside of this period.

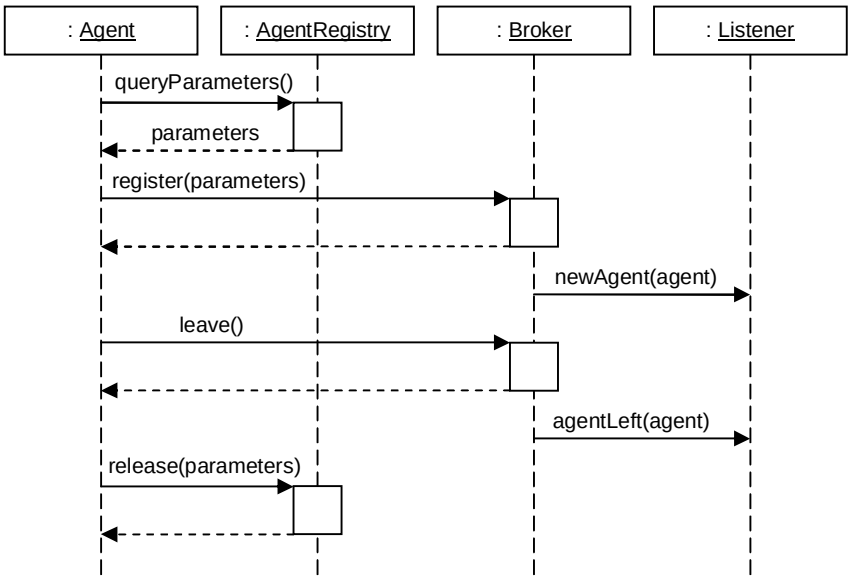

Figure 10. The typical lifespan of an agent in the framework. Note that the arrows do not represent method calls. Instead, the messages between agents are passed via network connections. However, the Broker does use local procedure calls to notify its listeners. The passed messages are coded in a language based on XML.

A sequence diagram is a graphical notation for describing the internal behaviour of a system. In object-oriented design, rectangles at the top row of a sequence diagram indicate objects, that is, instances of a class. Dashed lines extending downwards from these objects indicate the timeline. Discontinuities in the timeline are marked with pairs of diagonal lines (not shown in Figure 10). Arrows that are drawn between timelines represent messages that objects pass to each other. Usually these messages are method calls, as the objects are located in the same software environment. However, in the present framework these messages are coded in the XML-based language that the agents use to communicate. The pipe that relays the message is currently implemented as either a TCP or a UDP connection.

Before the agent registers to the Broker, it connects to a local service called an AgentRegistry. This service provides the agent with a unique identifier and a network address. These are required to register as a part of the system. The AgentRegistry service can also assign a Broker to an agent. This enables the system to automatically share the management of agents among multiple Brokers in a manner that evenly distributes the workload. This is a good method for meeting the efficiency requirements imposed on systems that consist of a large number of agents residing in the same environment. However, there is also another option available for a developer who wishes to distribute the management of a system. The developer can establish multiple local Brokers that agents directly register to. This might be a good choice when the system consists of many separate devices and a Broker can be assigned for each of them.

After the agent knows the parameters that are required to contact a Broker, it sends a registration message to the Broker. The message contains information about the agent and the filter encapsulated in it (see Figure 8 in Section 3.1.1.). The Broker stores this information for later use. The Broker also notifies all of its listeners about the new filter. Listeners are software components that have registered to receive notifications about changes to the software architecture of the system. Thus, listeners are comparable to Observers presented by Gamma and others [1994].

The registration of software components is done with a standard procedure call (a general mechanism in modern computer architectures and programming languages). The listeners are also notified of the associated architectural changes using this same method. Thus, the framework currently only supports listeners written in Java programming language, as the Broker was implemented using this language. Also, listeners have to be located in the same software environment as the Broker.

When an agent leaves the system, it notifies the Broker, which in turn notifies the listeners. Then, the agent contacts the AgentRegistry where it originally received the parameters required for registration. The AgentRegistry can subsequently release the unique parameters that were reserved for the agent and provide them for other agents that register later. The framework does not enforce agents to contact the registry service to release these parameters, but it is a good practice. However, the option to reserve parameters for later use has been left to the developer. This may be useful in some cases, as the software component that corresponds to the agent can still reside in some environment and continue to operate, although it is no longer a part of the system.

The Broker provides a central interface for the management of the system architecture. In the case that there are multiple brokers in a system, one of them manages the overall system. Other Brokers register to this central Broker as agents. As an example of the simplicity this promotes, Figure 11 presents a sequence diagram for a case where a visual editor is used to dynamically manage the system, that is, to change the architecture while the system is operational. This case also illustrates how connections are created between components.

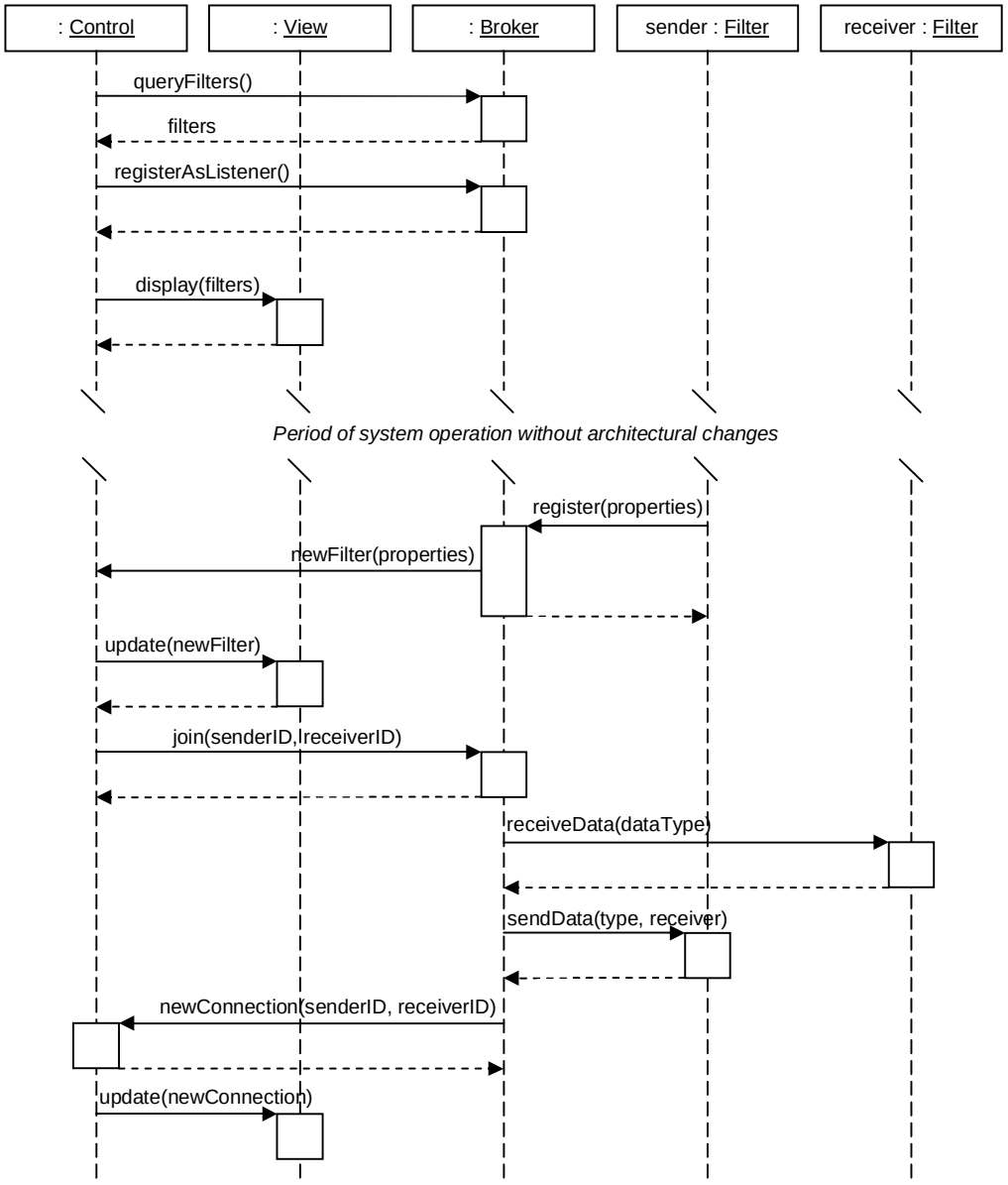

Figure 11. Sequence diagram of a case where system architecture is managed with a visual editor. The editor is split into two parts: the Control and the View components.

The visual editor application is divided into Control and View components for clarity of presentation. The Control component receives commands from the user interface and requests the Broker to make the relevant changes to the system architecture. The View component displays the components and is updated by the Control component when the architecture changes.

First, the Control component queries the current architecture from the Broker. Second, the Control component registers to the Broker as a listener in order to receive notifications when the system architecture changes. Then, the component updates the View according to the present architecture of the system. For a while, the system operates normally until a new agent registers to the Broker. The Broker notifies the listeners, including the Control component, about this event with a local procedure call. The described method complies with the Observer design pattern that is widely used in software development [Gamma *et al.*, 1994].

When the Control component receives the information about the new filter managed by the joining agent, it can update the View immediately in order to display the new filter in the user interface of the editor. At the same time, the Broker can inform the agent that its registration has been accepted.

In this example, these events are followed by a request from the Control component to connect two filters together, that is, to form a pipe. The originator of this request is the editor's user, who manipulates the architecture using a graphical user interface. The Broker reacts to this request by asking the agent that handles the receiving filter to prepare for incoming data. The agent accepts the connection by providing a Uniform Resource Locator (URL) that can be used to contact it. The agent also informs the Broker of the network protocols it prefers for connections, unless a protocol was already specified by the Broker. Next, the Broker asks the sender to connect to the recipient. When the sending filter has successfully formed a connection, it reports this success to the Broker. The Broker informs its listeners, including the Control component, of this new component. Finally, the View can be updated accordingly.

From the viewpoint of the visual editor's user, the filters are shown in some abstract graphical form, and they can be manipulated directly using the tools provided by the graphical user interface. For example, a connection between filters might be formed by dragging the sending filter over the receiving filter with the mouse and then releasing the mouse button. The graphically displayed architecture and the true architecture of the system are always equivalent, allowing for small delays. These slight delays are due to the time required for notifying all listeners about architectural changes and updating the visual representation.

The details concerning the connections (protocols, transmission rates, and so on) can be hidden from the user, although it is possible to allow them to be controlled when necessary. For example, a graphical dialog for changing these properties could be provided. Further, the dialog could be automatically generated based on the XML-based description of the filter. This description is provided by every agent when registering to a Broker.

The present implementation of the framework provides a starting point for developing psychophysiologically interactive computer systems. Next, two systems that were developed with the support of the framework are presented. The experiences from the development and the operation of these systems were used to evaluate the design patterns and solutions that were adopted for the framework.

3.2. System prototypes

In order to evaluate the support offered by the framework, it was used to construct two psychophysiologically interactive systems. The first system monitored the heart rate of a person with wireless electrocardiography (ECG). The general setup of the system is presented in Figure 12.

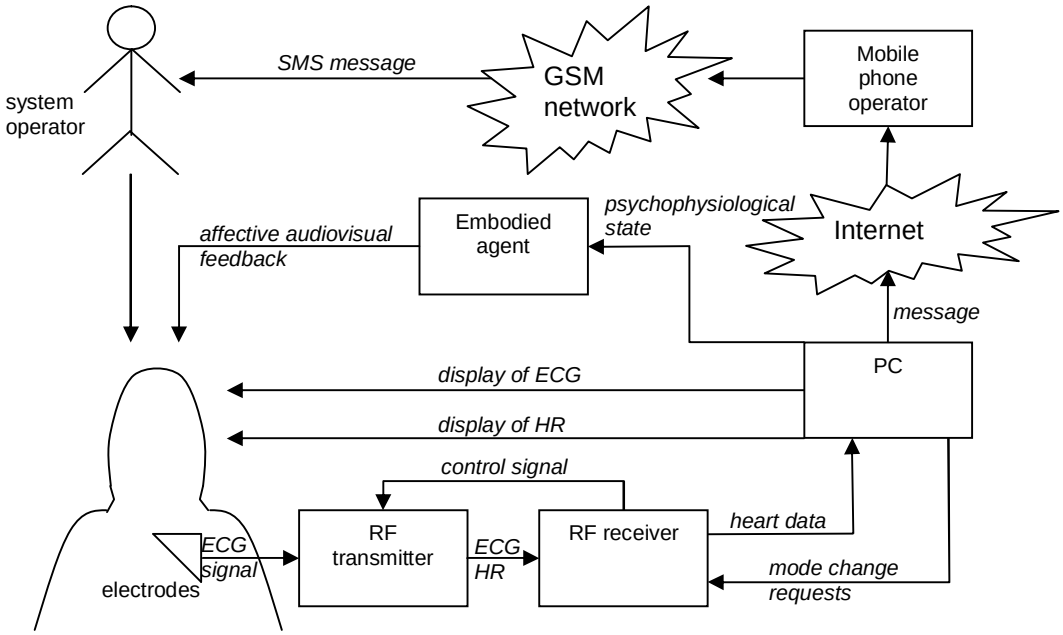

Figure 12. The setup for the remote heart rate monitoring system.

The ECG signal was registered with a wireless measurement system. The system was attached to the chest of the monitored person. The system acquired the ECG signal and provided either the acquired ECG values or heart rate it

derived from them, depending on its mode of operation. The data was sent via a radiofrequency (RF) connection to a RF receiver connected to a desktop personal computer (PC). The PC could request the electrode system to act in either of its two modes. The PC provided audiovisual feedback to the monitored person and a graphical interface for the operator of the system.

The graphical interface enabled the operator of the system to define thresholds for the heart rate. When the thresholds of a safe situation were exceeded, the system provided audiovisual feedback to the monitored person. The feedback was given by an embodied agent that instructed the person to relax and breathe calmly. Then, if the vital functions did not return to normal, but approached a critical state, the system alerted a system operator via Short Messaging Service (SMS). The SMS service was accessed through the Internet using the World Wide Web site of the mobile phone operator.

After the operator was contacted, the audiovisual agent informed the monitored person that human assistance was already on its way and assured that everything would be fine. When vital functions progressed towards normal, the agent gave positive encouragement and further instructions for relaxation to the monitored person.

The heart rate monitoring system was tested with four subjects. The system was applied to acted scenarios, where the monitored person induced heart rate changes with controlled breathing or parameters of the system were set to bias the assessment of signals towards abnormal functioning. This enabled a more controlled setting than actual clinical scenarios.

The framework was also used in setting up an experimental empirical study. In this study twenty-seven participants voluntarily activated either the *corrugator supercilii* (knits and lowers the brow) or the *zygomaticus major* (draws lip corners up) muscle. These muscles were monitored with facial electromyography (EMG). Voluntarily controlled facial muscle activity was held at one of three intensity levels at a time. Activations lasted for 30 seconds each. The power of the EMG was visually displayed to the participant throughout the experiment. The activity of the heart was registered with electrocardiography using the same wireless electrode system that was used in the first test setup. The empirical setup acted as a testing ground for the framework, as accurate timing and reliability of data collection was essential. Furthermore, multiple parallel signals (ECG and EMG) of differing characteristics and parameters of acquisition and processing were collected. Thus, the empirical setup was sufficiently different from the first heart rate monitoring system, in order to complement the results from the first trial of the framework.

3.3. Data acquisition

Electrocardiographic (ECG) data was acquired for both test setups using a wireless electrode system developed at the Tampere University of Technology [Vehkaoja and Lekkala, 2004]. The system is illustrated in Figure 13. Blue Sensor ECG electrodes were attached to a measurement patch that was connected to a radio frequency (RF) transmitter. The ECG electrodes were placed to the chest of the subject without skin preparation. The electrode system had two modes of operation. It either only acquired the raw ECG data or derived heart rate from the ECG. Then it delivered either the ECG or the heart rate data to the RF receiver. The receiver was connected to a desktop computer via a serial communications port. The sampling rate for ECG was 500 Hz and the recording passband was set from 0.07 to 192 Hz.

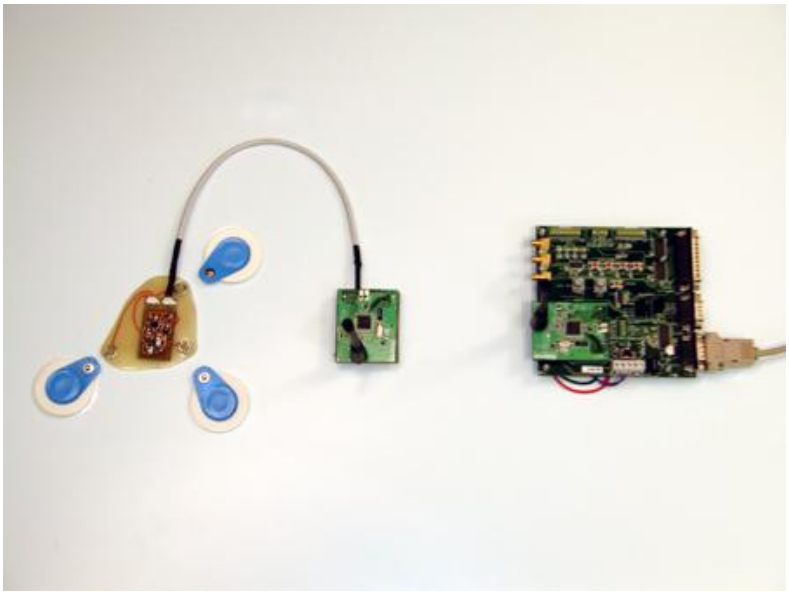

Figure 13. Wireless electrode system for electrocardiographic measurements.

For the second test setup, in addition to the ECG, facial electromyography was acquired with Grass® model 15™ differential amplifier using Ag/AgCl electrodes. Electrodes were placed above *zygomaticus major* and *corrugator supercilii* muscle sites on the left side of the face, according to the guidelines of Fridlund and Cacioppo [1986]. The skin was cleaned with ethanol and slightly abraded before electrodes were placed. The sampling rate was 2000 Hz and recording passband from 10 to 1000 Hz.

3.4. System architectures

The heart rate monitoring system provided audiovisual feedback to the monitored person and an interface for configuring the thresholds for normal heart rate. The software component responsible for the management of the graphical user interface was called the DemoFrame [Figure 14].

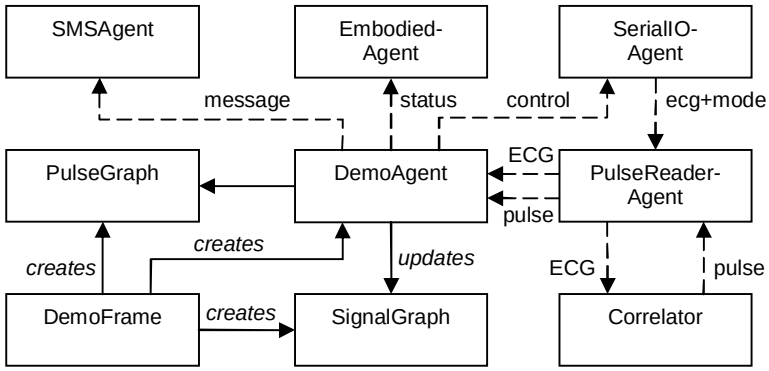

Figure 14. Software architecture of the heart rate monitoring system. Software components are presented as rectangles and connections as arrows. Dashed arrows are connections implemented as pipes of the framework. Regular arrows represent connections implemented using shared memory (i.e., not using the framework). Connections are labelled according to transferred data (in regular text) or the relationship between components (in italics).

The DemoFrame created components for displaying the ECG and the heart rate. These components were instances of the SignalGraph and the PulseGraph classes, respectively. The DemoFrame also had an intimate connection with the DemoAgent, which handled the interaction between the graphical user interface and the framework of software agents.

The signal from the wireless electrode hardware was read from the serial communications port by the SerialIOAgent. The SerialIOAgent managed the input from and output to the serial port. It delivered the received signal to the PulseReaderAgent. This agent transformed the signal into an ordinary ECG signal. Information about the mode of operation was encoded to the signal that the wireless electrode system provided. This information had to be removed before further analysis could be performed. Thus, the ECG signal could not be directly provided from the SerialIOAgent to the DemoAgent, but had to be first filtered by the PulseReaderAgent.

The PulseReaderAgent employed another agent, called the Correlator, to compute the correlation between the ECG and a signal template in order to

recognize heart cycles. Based on the correlation, the Correlator derived the heart rate and delivered it to the PulseReaderAgent, which in turn provided it and the original signal data to the DemoAgent. The DemoAgent used the received data and its internal state to decide when the mode of operation had to be changed. When the agent decided to change the mode, it first told the SerialIOAgent to relay a command for mode change to the wireless electrodes via the serial port connection. Then, it instructed an embodied agent about the new status and told the SMSAgent to send a message to the monitoring personnel, if necessary. The embodied agent provided the appropriate audiovisual feedback to the monitored person.

The software architecture used in the setup for the empirical study shared many software components with the previous system [Figure 15]. This could be expected, of course, as both systems registered and processed electrocardiographic data. However, it was not obvious that the same components could be used without modifications.

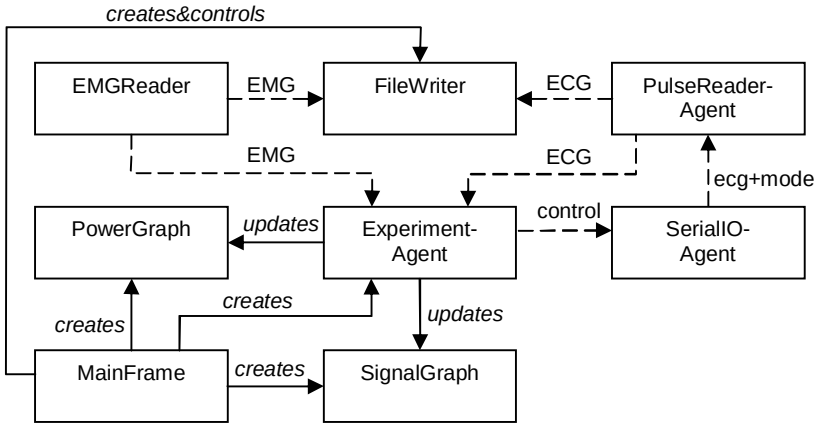

Figure 15. Software architecture of the system used in the empirical study.

As the experimental tasks consisted of voluntary activations of facial muscles, the system provided the participant visual feedback on his or her facial activity via a graphical user interface. The software component responsible for managing the graphical user interface was called the MainFrame. The MainFrame created components for displaying the raw EMG and ECG signals and the mean power of EMG. The component that displayed the raw signals was called the SignalGraph. The SignalGraph component was developed for the previous system and reused in the construction of this second system. The mean power of EMG was displayed to the participant using the PowerGraph component. The MainFrame also created and communicated with the ExperimentAgent, which handled the interaction

between the graphical user interface and the framework of software agents. Thus, its task was very similar to the operation of the DemoAgent, which was a part of the first test setup.

Wireless electrodes were joined to the system using the components that were developed for the first system (see Figure 14). SerialIOAgent controlled and acquired data from the wireless electrodes via a serial port. The PulseReaderAgent converted the coded signal into raw ECG data, which was sent both to the ExperimentAgent and the FileWriter, which stored the collected data for off-line analysis.

The ExperimentAgent monitored the mode of the electrodes and sent a request to change mode, if electrodes were providing the system with the computed heart rate instead of the raw ECG signal (as shown in Figure 12). EMG signal was acquired by a separate program that sent the obtained sample values as UDP multicast packages. These packages were received by the EMGReader agent, which provided the EMG data for the ExperimentAgent and the FileWriter. ExperimentAgent updated the respective displays with the data and, when an experimental task was to be performed, changed the mode of operation when the activity was within the required target range.

4. Results

4.1. System construction

The present framework implemented the Pipes and Filters, the Mediator, and the Observer design patterns. Table 5 summarizes the implications of these patterns. The general implications were derived from the theoretical qualities of the design patterns and they are common to all implementations of these patterns. These general implications are illustrated by the specific implications that were acquired from testing the framework. Some of these implications are more evident when results from the operation of the two systems are inspected. These results will be presented in the next section.

The Pipes and Filters pattern removed the need for temporarily storing data, for example, into files. However, the pattern still allowed intermediate data to be inspected. This benefit was evident especially when implementing the software for the empirical test setup. Both raw electromyographic data (i.e., intermediate data) and events derived from it (i.e., the result of processing) could be effortlessly recorded to a file, although the latter data was available at a later processing stage than the former.

Another benefit from using the Pipes and Filters pattern was the flexibility provided by simple exchange of filters. The simple interface common to all filters enabled them to be replaced with other filters that performed the same task, without modifying other parts of the system. When developing the second test setup, an agent that provided simulated electromyographic (EMG) data was implemented for testing purposes. This agent was used to simulate the operation of the system when the hardware for EMG acquisition was not available. Replacing the agent with another agent that actually acquired data from a subject was a straightforward operation due to the flexibility of the design pattern.

The possibility to recombine existing filters and construct new systems from existing components was also supported by the use of the Pipes and Filters pattern. This characteristic relieved the effort when constructing the second test system as filters from the first system could be reused without any extra effort.

The method that was used in implementing the Pipes and Filters pattern also supported the extensibility of systems. In the first test setup, the SMSAgent and the EmbodiedAgent were not a part of the agent architecture, that is, they were not developed using the present framework. However, integrating them

Table 5. Summary of design patterns used in the framework and their implications.

	General implications	Specific implications
Pipes and Filters [Buschmann *et al.*, 1996]	Removed the need to temporarily store the data in one location. Enabled intermediate data to be inspected.	The second setup could effortlessly record data from different stages of processing to the same file.
	Parts of systems could be exchanged without effort.	The second setup could be reliably tested by exchanging the components for data acquisition with components for data simulation.
	Parts of systems could be reused.	The second setup could use parts of the first one.
	Architectures were adaptive as a consequence of the exchangeability and the reusability of components.	Agents could be employed in arbitrary order at arbitrary times.
	Many types of processing architectures were supported.	Communication between different stages of processing was possible in both setups.
	Efficiency was supported by the possibility to share the common components of pipelines.	Common tasks had to be performed only once during the operation of the two setups.
	Components were active in processing the data.	There were no idle periods during the operation of the two systems.
Mediator [Gamma *et al.*, 1994]	Mediator stored behavior that would otherwise have been distributed.	The Broker provided a simple method and a central interface for defining the architectures of the two systems.
	Software components did not have to directly refer to each other.	The reused components from the first setup did not have to be modified in order to be joined to the second system.
	Interactions between individual software components were replaced with interactions between them and the Mediator.	Different levels of coupling could be separated when constructing the first setup. This simplified its construction.
Observer [Gamma *et al.*, 1994]	A uniform method could be used for communicating between components.	In the case of the two setups, no agent-specific communication had to implemented.

to the system was simple due to the implementation of the connections (i.e., pipes) between agents. The connections were managed with TCP and UDP, which are common protocols that are interoperable between several environments and supported by many libraries of program components. For example, the SMSAgent was implemented in Python programming language. Integrating the SMSAgent was supported by network functionality in the standard Python library (i.e., the library included with the distribution of Python).

The Mediator design pattern was employed in the present framework to manage changes to the system architecture. Each agent registered to a Broker and all changes concerning the agent were handled by that Broker. Thus, using the Mediator pattern kept agents from referring to each other explicitly, which had several benefits.

First, the Broker stored the behavior that would otherwise have been distributed. In other words, the Broker kept the architecture of the system consistent with the overall behavior that is required from the system as a whole. In the construction of the two test setups, this meant that the individual filters were joined into an operational system through the Broker. Thus, the Broker provided a more centralized and manageable method for defining the system architecture than defining references to other filters separately for every filter.

Another benefit was the decoupling of filters, that is, the removal of direct references between filters. As a consequence, the architecture of systems could be managed in a uniform manner, regardless of the actual filters that formed these architectures. There was no need to modify the filters that were reused from the first test setup to the second system. The present framework enabled the filters to be reused simply by modifying the architecture defined within the Broker.

Finally, the interactions between individual filters were replaced with interactions between the filters and the Broker. For example, the DemoAgent in the heart rate monitoring system provided data for the SMSAgent, the EmbodiedAgent, and the SerialIOAgent (see Figure 14 in Section 3.4). The DemoAgent received data from the PulseReaderAgent. In addition to these agents, the DemoAgent also provided and received data from components of the graphical user interface, but these connections were not managed by the Broker. Separating these two types of connections simplified the management of the relationships between components. As the DemoFrame agent was responsible for updating the graphical representation of received data and handling user input, the coupling between the agent and the graphical

components was tight. Thus, the Broker did not have to take care of the architecture associated with interacting with the user.

The quantity of connections was also reduced by removing the interactions between individual filters. The filters did not have to directly know the presence and state of any other agent besides the Broker. All of the other agents were accessed through the Broker. This benefit was augmented further by the Observer pattern. Any component could obligate the Broker to inform it of changes to the architecture of the system. The following positive results were obtained for using the Observer pattern.

In both test systems, a separate component for management of the architecture was created and registered to the Broker. This component received notifications each time an agent registered to the system. When all necessary agents were available, the component asked the Broker to compose the system architecture. Thus, the time when an individual agent registered to the Broker could vary without affecting the construction of the system. Also, different architectures could be rapidly prototyped through the development by changing the defined architecture only within one component.

The Observer pattern also enabled a uniform method to be used for communicating between agents. As a result, the Broker did not have to know how any of the agents had been implemented. In spite of this, it could efficiently communicate with all of them. As a consequence, both test setups could be constructed without making any changes to the communication methods, although components exchanged data at varying levels of abstraction.

In addition to the results that were presented, it can be argued that the constructed systems solved the challenges of psychophysiologically interactive computer systems. These solutions and the challenges that they addressed are summarized in Table 6.

Table 6. Challenges for psychophysiologically interactive computers and the solutions adopted in the framework (numbered according to Section 2.2.).

Challenge	Solution
1. Psychophysiological data is context-dependent.	The Broker provided a context by storing information about the registered filters. The context could be inferred from the properties of these filters. The services of a dedicated context agent could also be automatically provided for agents that required information about the context.
2. Parameters of data acquisition must be known in analysis.	The XML-based description of filter capabilities was capable of handling multiple levels of abstraction, including details of data acquisition. Also, information of all preceding processing stages could be preserved if the description was layered (i.e., preceding stages were encapsulated within succeeding ones).
3. Psychophysiological data is non-specific.	The fusion of parallel physiological signals is an effective method for resolving ambiguities. The framework supported multiple input channels for every filter, which enabled this fusion.
4. Psychophysiological responses vary between individuals.	Information about the individual could be included both to the XML-description of data and to a separate storage (see the solution of the first challenge).
5. Recognition of psychophysiological events is unreliable.	Both the fusion of parallel signals and information about the context could help in resolving ambiguities. See the solutions to the first and third challenge.
6. Different domains of data and analysis must be supported.	The framework did not restrict the nature of the data that the filters processed. The domain could be included to the XML-based description of filter properties. For example, the description could contain information about the placement of electrodes, enabling spatial analysis.
7. Systems are often distributed.	Placing different filters in different environments did not require any extra effort. The pipes and the communication between agents were implemented in a cross-platform manner.
8. Systems must be context-aware and adaptable.	The Broker provided a central interface to the whole system architecture and it could be requested to notify when the architecture changed. However, reacting to these changes was left to the developer.
9. Support for long-term monitoring must be included.	Context changed during long-term monitoring. Context-awareness and adaptability of the architecture supported long-term monitoring.
10. Different levels of data processing and communication should be supported.	The pipes-and-filters architecture was versatile and enabled splitting and merging the processing flow at any point of processing.

4.2. System operation

In addition to the support that the framework provided for the construction of the test setups, its value is evident in their reliable and robust operation. The operation of the first system was tested in several scenarios in order to find out if the collaboration of the agents would result in rational operation of the whole system. The scenarios were different combinations of succeeding events. The events and the responses of the system are presented in Table 7.

Table 7. Events and responses from testing the heart rate monitoring system.

Event	Response
Heart rate exceeded normal, but remained less than critical.	The embodied agent instructed the person to relax and breathe calmly.
Heart rate continued to accelerate and became critical.	The DemoAgent instructed the SMSAgent to send a message to the health care personnel. The message contained the relevant medical history of the patient. At the same time, the embodied agent informed the person that medical personnel had been contacted and continued to calm him or her down.
Heart rate decelerated to less than critical.	The embodied agent told the patient that relaxation was working and gave further instructions for relaxation.
Heart rate returned to normal.	The embodied agent told the patient that the heart rate was again normal, but suggested that it would be better to avoid further stress for a while.

The system responded consistently to each event, whether it was due to heart rate changes induced by voluntary breathing patterns or by parameters being adjusted to trigger the event. It was possible to adjust the parameters so that an event was only produced by voluntarily induced heart rate changes.

Figure 16 shows the operation of the heart rate monitoring system as a sequence diagram. The PulseReaderAgent constantly provided heart rate data to the DemoAgent. The DemoAgent decided when the situation was no longer normal and informed the EmbodiedAgent accordingly. The EmbodiedAgent provided the appropriate feedback, depending on the current and previous states of physiological functioning.

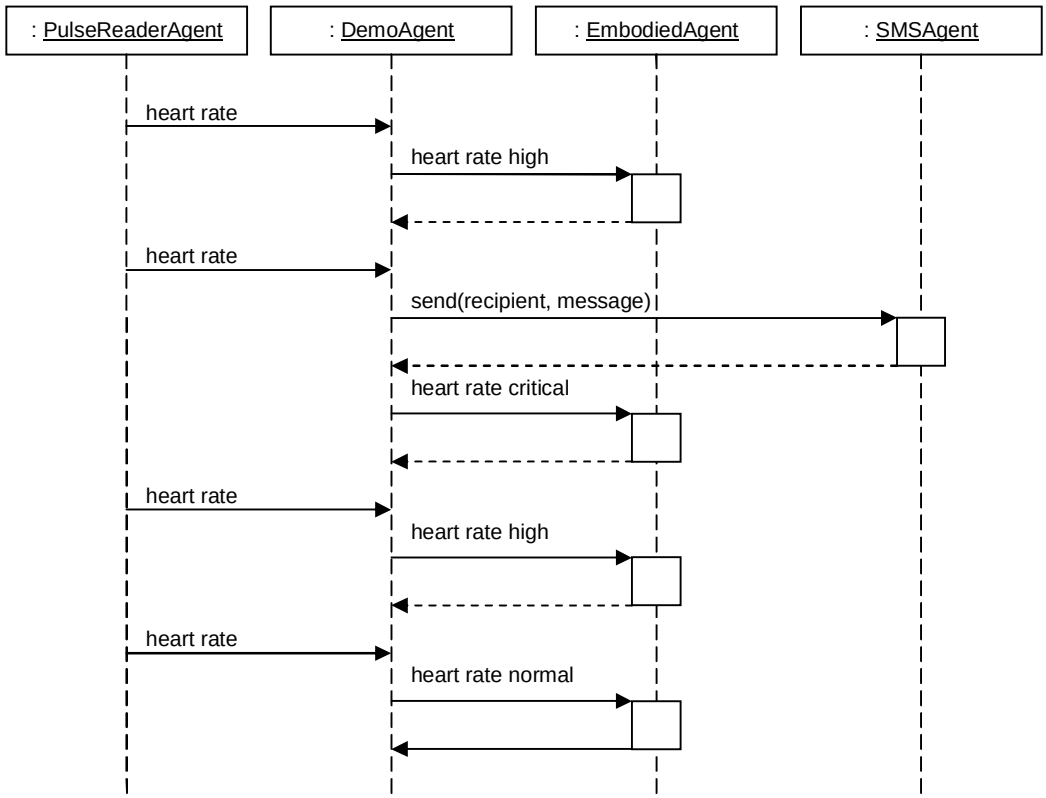

Figure 16. The operation of the heart rate monitoring system.

If the heart functioning did not normalize and situation was assessed critical, the DemoAgent sent an alarm to health care personnel through the SMSAgent. Then, the DemoAgent informed the EmbodiedAgent of this, and it provided the appropriate feedback, which included assurance that help was already on its way. The EmbodiedAgent was also informed when the heart functioning was no longer critical and when it returned to normal.

The second system was required to handle a larger amount of data than the first test setup, as both electromyographic (EMG) and electrocardiographic (ECG) data were registered. In addition, markers for the onset and offset of each experimental task were collected as well. The timing error in these markers was at most 25 milliseconds, which was more than acceptable considering that the parameters of interest (i.e., heart rate and heart rate variability) were inspected and analyzed during time periods of 30 seconds.

Although there were no architectural changes to be made during its operation, the second system was required to adapt according to the EMG signal power and the heart signal data it received. The mode of the operation had to be changed when the power of the EMG signal reached the required target level of activation intensity during a task. The system started an internal

timer in order to end the task after certain period of activity. Also, the type of signal that the wireless electrodes provided could require the system to take the initiative. If heart rate data was received instead of raw ECG signal, the ExperimentAgent requested the raw signal from the ECG electrodes.

However, these adaptations could be performed by individual agents. Thus, there were no specific events triggered and no associated messages to be passed to other agents, as was the case in the first test setup. Consequently, visualizing the operation of the system with sequence diagrams (see Figure 16) would not clarify the described behavior.

In addition to the results that were specific to one of the systems, there was some generic support that the framework provided for the operation of both systems (see Table 5). First, the chosen implementation of the Pipes and Filters pattern enabled the construction of recursive processing architectures. Filters that were placed at later stages of the processing pipeline could provide feedback to previous filters.

As an example of the benefits of these kinds of architectures, the DemoAgent of the first system could request the mode of the wireless electrode system to be changed. This request was sent to the SerialIOAgent that provided the heart signal. Events were extracted from the heart signal and the DemoAgent used these events to decide when the mode should be changed. The communication between the ExperimentAgent and the SerialIOAgent in the second test setup was performed in an equivalent manner. The two collaborating agents were situated at different levels of the processing flow and the latter agent dealt with relatively high-level data. Nonetheless, the two agents could communicate due to the support that the framework offered for abstract communication. This support was the result of a number of solutions that were adopted into the framework. These solutions included the method that was used to implement the pipes, the abstract and extensible language for inter-agent communication, the use of the Mediator pattern, and the use of the Observer pattern [Buschmann *et al.*, 1996; Gamma *et al.*, 1994].

Second, the efficiency of systems was promoted by implementing the common parts of processing as filters. Due to the chosen implementation for the pipes and the filters, these filters could be used in several parallel processing flows. For example, in the empirical setup the PulseReaderAgent provided the electrocardiographic (ECG) signal both to the FileWriter and the ExperimentAgent. Thus, the extraction of ECG signal from the encoded signal provided by the wireless electrodes had to be performed only once. As another example, the system used in the empirical study contained a filter that transformed electromyographic (EMG) signal into activation intensity (i.e.,

signal power) data. The filter provided this data to three software components. One of these components showed the data to the person whose facial activity was being monitored. Another component stored the data into a log file and the third component monitored the intensity level of activity. Depending on the level of activity and elapsed time, this component initiated and ended experimental tasks. In this example, it was necessary to compute the power only once. All three components could use this data without converting or otherwise modifying it, saving processing resources of the system.

Third, the operation of systems was flexible due to the implementation of the filters. The filters were active, that is, they actively received and sent data whenever a sufficient amount was available [Buschmann *et al.*, 1996]. In other words, there were no idle periods in the operation of systems. This resulted in the system responding rapidly to changes in the physiological parameters. Actually, the reported 25 millisecond timing error in the second setup was due to the intrinsic limitations in the acquisition of the electromyographic data, instead of limitations imposed by the framework.

Finally, the adaptability of system architecture was evident when the operation of the two setups was at its beginning. The operation of a system was started by putting the Broker into operation. Then, the operation of individual agents was started in an arbitrary order. In other words, the system was operational before agents registered to the Broker, although the system did not perform any functionality until a suitable set of agents was available.

5. Discussion

This thesis presented a framework for the construction of psychophysiologically interactive computer systems. The framework was created in order to address challenges developers face due to both the nature of psychophysiological signals and the special characteristics of psychophysiological human-computer interaction. The framework was tested by constructing two different systems with it. The first system was developed for remote heart rate monitoring with a wireless electrode system. The second system was used in an empirical study and registered both electrocardiographic and electromyographic data.

The results showed that the framework supported the construction and operation of these two different systems. The heart rate monitoring system provided direct feedback to the monitored person, adapted its operation according to extracted events, and contacted other persons when it assessed the situation as hazardous. Thus, it covered the three classes of psychophysiological human-computer interaction that were presented in this thesis: biofeedback-based, biocybernetically adaptive and extended interaction. The setup for the empirical study supported both biofeedback-based and biocybernetically adaptive interaction. The former was implemented by displaying the electromyographic data to the monitored person. The latter was supported in order to change the experimental task after a certain period of controlled activity.

The level and amount of processed data also varied within and between the systems. The first test setup registered only heart related data and based its actions on present activity only. The second setup registered the activity of both the heart and the facial muscles. It also derived the power of facial activations from electromyographic data. The derivation of this measure required a longer period of time compared to the derivation of the heart rate performed by the first system.

The second system also required support for multiple levels of data abstraction. The EMG and the ECG data were acquired by the system as individual samples. They were later processed to derive the heart rate and the power of the EMG. The first system also handled high-level data, as information of the medical history of a (virtual) patient was contained in the SMS messages at a relatively high level of abstraction.

The contexts in which the two test setups operated were stable. Thus, there was no need to adapt their architectures once a system was constructed. Nonetheless, the systems did provide some preliminary results that indicate

that the framework does support the run-time adaptation of the system architecture and that this adaptation can be managed through a single interface (i.e., the Broker). In both systems, agents were registered to a Broker in an arbitrary order. As soon as the suitable set of agents was available, a software component was informed and it instructed the Broker to construct the architecture. Thus, the system was operational during this whole period of registration, although it did not yet perform any external task. Internally, the system was active and sensitive to changes in the software environment, that is, the availability of components.

In any case, the coverage of the two test setups was limited, when the adaptability of systems is considered. There was no need to reconfigure the architecture once it had been defined. Thus, although the two test cases brought out only positive results from the use of the three design patterns, there are also potential drawbacks that have to be considered especially in the future development of the framework and psychophysiologically interactive computer systems.

First of all, the Pipes and Filters design pattern that was used in the framework is not efficient if a large amount of global data has to be shared [Buschmann *et al.*, 1996]. Considering the software architectures of psychophysiologically interactive computer systems, contextual information could be considered global data. Generally speaking, the context is an important source of information in the analysis of psychophysiological data [Cacioppo *et al.*, 2000]. However, the context is mainly significant as an additional source of information for data fusion. Thus, earlier processing stages do not require information about the context, which relieves the efficiency requirements posed on the sharing of contextual information. This assessment is supported by the empirical results from comparing feature-level fusion with semantic-level fusion that were presented in Section 2.2 [Busso, *et al*., 2004; Zeng *et al.*, 2004].

Other global data that is significant for psychophysiological human-computer interaction consists of data acquisition parameters, such as sampling rates and electrode locations. These parameters affect the operation of the associated filters. For example, analysis of heart rate variability requires a larger number of samples at a time than measuring the mean amplitude of the raw ECG signal. Due to the varying requirements of filters, the rate at which the information flows through a pipe cannot be defined by the producer of the data, that is, the filter at the transmitting end of a pipe.

One solution to this problem would be to have a separate pipe for every connection between two filters. Each of these pipes would provide data for the

receiver at the rate that it processes it. However, this would be inefficient, as storing copies of or references to the same data items would increase the required memory and processing capabilities. Another possibility would be to upkeep a common history buffer for each filter that stores the processing items that the filter produces [Ilmonen and Kontkanen, 2003]. This would require that the receivers are active in reading the buffer. When distributed systems are considered, querying new processing items would have to be performed either with a high-level language or by executing remote procedure calls, both of which add a significant overhead and are tied to a particular software environment.

The present implementation of the framework employs buffers in order to take care of receiving and sending processing items through pipes. The developer retrieves incoming processing items from a buffer that is a part of the filter itself. The results from processing are placed in another buffer and sent when a sufficient amount of data is available. Each filter is responsible for flushing these outbound buffers when they are full. The receiving filters are active, that is, they read data whenever it is available [Buschmann et al., 1996]. This reduces the time required for the system to respond, compared to systems that use passive filters. Passive filters act only when forced by external stimuli.

As a result of implementing active filters, the filter that produces the data controls the rate at which it is sent. This requires that the filter is informed about the requirements of its recipients. Also, the capabilities of the transmission channel (i.e., pipe) have to be taken into consideration and thus also available for a sender. Although informing every component that sends data of these parameters requires an investment of processing resources, the parameters are not subject to frequent changes. Thus, information about the components can be stored into one central location, where it is available to every component. This enables the system to rapidly react to changes in hardware and software environment.

In the present framework, the Broker acts as the storage for these parameters. It upkeeps a list of all filters that form the system. This makes it possible to evaluate and compare the possible compositions of pipes and filters (i.e., architectures). Other components can register to the Broker as listeners in order to receive notifications when the environment changes in some way. Thus, the Broker implements the Observer design pattern as it stores and provides global data for the whole system [Gamma et al., 1994].

If this global data is seen as the context, the role of the Broker is similar to the context-manager in the Embassi system, the discourse memory in the multimodal SmartKom system, and the dialog manager in the work of Flippo

and others [Elting et al., 2003; Reithinger et al., 2003; Flippo et al., 2003]. These systems focus on multimodal dialogue for controlling individual applications. The context that these systems manage consists of different hardware devices and software components. In this sense, they are comparable to psychophysiologically interactive computer systems. From the results that have been acquired from these systems, it seems that having a single component that provides context is efficient and easy to manage.

However, due to the complexity of psychophysiological phenomena, the analysis of psychophysiological data might need additional contextual information. This information might concern the time of day, the physiological characteristics of a person, and the temperature, for example (see, e.g., [Ward and Marsden, 2003]). This can quickly add up to an unmanageable amount of information, which is why other multimodal systems are not directly comparable to psychophysiologically interactive computer systems. The agent-based architecture of the present framework enables the management of context to be distributed among multiple agents. This might be necessary in large-scale systems. Thus, one way in which the framework supports the future development of systems is by enabling the development of agents that manage some specific sub-context.

Currently, only components that reside in the same software environment can request notifications from a Broker. However, in some cases components have to be informed of changes to global parameters, regardless of the environment where these parameters are stored. As an example, a Broker dedicated to handling a person's digital calendar would have to inform remote agents that require information of the person's meeting schedule in order to minimize interruptions.

This is why remote listeners have to be implemented in the future development of the framework. The language used for inter-agent communication provides support for adding this new functionality. Due to the extensibility of the language, new message types can be defined without compromising the operability of existing components. The language also enables the description of an agent's output channel to incorporate the earlier processing stages. This way, the data acquisition parameters can be preserved through the whole processing pipeline.

Another possible negative consequence from the use of the Pipes and Filters pattern could be a large overhead in processing due to data transformations [Buschmann et al., 1996]. These transformations are the result of using a single data type for all filter input and output. However, when using the present framework, it is easy to avoid this drawback. Support for exchanging data was

implemented for only the most basic data type, that is, a byte. Other data types can be implemented on top of this type, preserving the support for all types of data. Actually, most psychophysiological data consists of discrete numeric values that can be readily handled as bytes. Furthermore, the communication between components is also supported by the XML-based description they provide when registering to the system. The description of a filter can specify the type of data it provides or accepts as input. This helps the system to evaluate which filters can interact with each other.

The presented results from the experimental setup demonstrated this support for the interoperability of components. In this setup, ECG and EMG power data were provided to multiple components. These components could readily handle the data without performing any preprocessing on the data, including data transformations. In addition to this finding, the experimental setup also demonstrated that exchanging filters was simple due to this interoperability. The component that provided data from the EMG acquisition equipment could be replaced with a component that provided simulated EMG data. This finding also supports the notion that the Pipes and Filters design pattern enables the rapid prototyping of systems, as proposed by Buschmann and others [1996].

In addition to the possible overhead that might be imposed by data transformations, also the chosen method of inter-agent communication may hinder performance. In the current implementation of the framework the connections between agents are managed with the TCP and the UDP protocols. Despite their advantages, the TCP and UDP protocols may hinder the performance of components that are located in the same software environment. These protocols require data to be packaged prior to transmitting it via a network connection. This packaging imposes a small overhead for transmitting data. It would be easy to add support for other types of connections in addition to TCP and UDP streams. These connections could include those native in Unix variants (see Section 2.3) or connections based on shared memory. Shared memory could be used to construct pipelines, if the filters shared the same environment. This would eliminate the overhead from the network protocol including, for example, the construction of packages and the necessity to buffer processing items before sending them through a pipe.

However, the TCP and UDP protocols are common protocols that support interoperability between environments and this interoperability supports the construction of distributed systems. These protocols are also supported by many libraries of software components. As an example, the integration of the SMSAgent with the rest of the system was supported by network functionality

in the standard Python library (i.e., the library included with the distribution of Python). Thus, it seems that no single protocol is superior compared to other protocols. Several protocols for inter-component communication should be supported in order to provide a suitable method for each case, depending on where components are located. It should be noted that both components should be able to handle the method. Thus, components that support methods that are not commonly supported will have difficulties in connecting with other agents.

Actually, interoperability between environments was assessed to be an important characteristic for psychophysiologically interactive computer systems that are often wearable, mobile, and distributed. Thus, the support offered for distributability was why TCP and UDP protocols were chosen as the first implemented inter-component communication method. This was also one of the main reasons for deciding to use the Pipes and Filters design pattern in the framework [Buschmann *et al.*, 1996]. However, the resulting distributability and efficiency could not be conclusively verified in the present work as both test systems were run on a single computer due to their small scale.

The basic Pipes and Filters pattern enables only linear processing flows. However, in the present work this pattern was extended to handle more complex processing flows. Pipes were implemented using buffers that were specific for each filter. This enabled the construction of systems that have several parallel and overlapping processing flows. Also processing flows that form cycles were possible. The ability to split processing flows enabled the common parts of algorithms to be implemented as shared filters and computed only once. The ability to form cycles supported the adaptability of systems, as feedback from later stages of processing could be provided to earlier stages. This enabled the two test setups to request certain type of signal from the wireless ECG measurement system, for instance. This functionality could not have been readily implemented using previous tools, such as the PhysioToolkit [2003]. These tools do not provide a method for providing feedback to earlier processing stages and neither does the UNIX shell that is used in executing these tools.

Although the framework does seem, based on the previous discussion, to address the potential drawbacks of the Pipes and Filters design pattern, other architectures could be more appropriate for some psychophysiologically interactive systems. For instance, Buschmann and others [1996] suggested that if the drawbacks associated with the Pipes and Filters design pattern are assumed to pose great risks to a system, the Layers pattern could be considered as a replacement. The Layers pattern simply arranges the components to

different levels of abstraction. In the case of psychophysiologically interactive systems, this would require the use of feature-level fusion. However, feature-level is not a suitable method for all signal processing. In addition, the reuse of different components (e.g., implementing common parts of algorithms) is limited to the same processing layer. If the system has many common parts, the layers are an inefficient architecture for it.

Nonetheless, the present framework does offer a method for arranging the filters into different layers, if the developer should so desire. Different layers can be grouped under their respective Brokers. Then, these Brokers should register to a central Broker that manages the architecture as a whole. Actually, it is possible to add a second layer or an arbitrary number of layers by grouping layers together, under a yet another Broker. Thus, the Layer pattern can be implemented without extending or modifying the existing framework. This way, the system can gain the advantages offered by the Layer pattern, while preserving some of the flexibility and efficiency offered by the Pipes and Filters pattern.

For example, in the present framework they were implemented using a method that enabled more complex processing flows in addition to linear ones, as was previously discussed. If the filters forming these flows were grouped into layers, the structures of these flows would be preserved. The layers would only add an additional level of abstraction.

In addition to the Pipes and Filters pattern, the other design patterns in the framework have potential drawbacks that have to be considered. The framework uses the Broker as a Mediator, which might increase its complexity and consequently decrease its maintainability in the future [Gamma *et al.*, 1994]. This will happen if the communication between agents becomes increasingly complex. To prevent this, it is essential to limit the extent of the services offered by the Broker to a bare minimum. For example, it would be tempting to include services for the automatic configuration of systems to the Broker. In the present framework, the Broker is a passive mediator that connects components only when explicitly requested. This keeps the Broker simple and maintainable. The automatic configuration, on the other hand, can be performed by a separate agent. Thus, the future development of the framework should focus on extending it with new agents, instead of adding functionalities to the existing ones.

The Broker is also a part of the Observer pattern, performing the role of the Subject. A Subject notifies Observers that have subscribed to it in order to receive notifications when the Subject changes. In the present framework, the Broker manages the architecture and notifies registered components when the

architecture changes. The potential drawback of this pattern is that components are unaware of each other's presence and might request changes that are costly for other components. This can be avoided by assigning only one component to modify the architecture through the Broker. This suggests that the present support for local Observers is sufficient and supporting remote listeners is actually undesirable.

However, remote listeners might be required for efficient management of distributed systems. For example, as previously argued, it is necessary to share contextual and other global data in order to process psychophysiological data. Remote listeners are an intuitive method for keeping track of changes to this data. As a more concrete example, the physiology of a person might be monitored at a remote location. The acquired measures could be shown on a mobile device that the operator of the system carries with her. If the operator would then notice clues indicating that one of the measures is giving spurious data, she could request other measures to be collected to confirm this observation. This would require the remote components to communicate with the Broker in order to modify the architecture accordingly. Both the part of the system that monitors the signals and the part of the system that displays them on the mobile device would have to be modified.

On the other hand, managing a large distributed system from one central location (i.e., Broker) might be complicated. Keeping the information about components up to date requires that all components have the means to communicate with the Broker, which might be located in a completely different remote environment. Furthermore, the Broker might form a bottleneck if the architecture consists of a large number of filters and architectural changes are committed very often [Moran *et al.*, 1998].

Although supporting the use of several brokers was not a specific concern when designing the framework, the framework does enable their use. Several brokers can be employed using a similar method as when implementing the Layers design pattern, as previously was described. First, the system is divided into smaller manageable subsystems, all of which are managed by a separate broker. Then, the brokers that manage subsystems are registered to one central broker as filters. This way, the management of the system can be distributed, but there is always one central interface for managing and inspecting the system architecture (i.e., a central Broker). Furthermore, if support for remote listeners is added to the framework, the cost of changing the architecture can be controlled by permitting only other Brokers to commit changes to the whole architecture.

It should be noted that the manager of the architecture does not have to be human, as was the case in the two test setups. A software agent could take the place of the system's operator, replacing human input with its own goal-oriented decisions. The goal of the agent would be to find an architecture that is optimal according to some criteria. It can be argued that management should actually be automatic when proactive systems are considered. Humans should only supervise the actions of a system, instead of constantly interacting with the system [Tennenhouse, 2000]. Proactive systems can make (architectural) decisions at a rate much faster than human judgement. This is why requiring constant human input would hinder the performance of a system.

Furthermore, as discussed in the introductory chapter of this thesis, psychophysiologically interactive computer systems have a natural disposition towards proactivity. Thus, it is desirable to design architectures that retain the advantages gained from the characteristics of proactive computing. One of these characteristics is the ubiquity of computing [Tennenhouse, 2000]. Ubiquitous computing supports human activity by interrupting the person as seldom as possible [Weiser, 1993]. Explicit management of the system would require the user to constantly interrupt the task or focus on the architecture and the task simultaneously, limiting the system's applicability to ubiquitous tasks. Of course, this distraction could be eliminated by dedicating another person to managing the system. However, this would be unnecessary waste of human resources.

In conclusion, it was shown in the present thesis that the construction of psychophysiologically interactive computer systems could be supported with the implemented framework. The results from two test setups showed that the framework enabled systems to be constructed of software components that operated in different environments. The framework also promoted the reusability of software components and supported signal processing that did not depend on the abstraction level of the data. The systems created with the framework were extensible, due to the ability to effortlessly join new components to existing system and the possibility to extend the framework itself. Thus, it was shown that using the framework reduces the effort of creating robust computing applications that utilize physiological data. Furthermore, software applications that are constructed with the framework can evolve from prototype systems to real-word applications, as the framework relieves the possible drawbacks of the chosen architectures and the design patterns implemented within them.

6. Summary

This thesis presented a software framework to support the construction of psychophysiologically interactive computer systems. Psychophysiologically interactive systems collect physiological signals and extract psychophysiological measures to be used in human-computer interaction. The extracted measures can be used to select appropriate feedback to the monitored person and adapt the operation of the system.

Psychophysiological human-computer interaction faces many challenges due to the diversity of its applications, the characteristics of psychophysiological signals, and the complexity of psychophysiological phenomena. When designing the framework, several applications were inspected in order to identify these challenges. After these challenges were identified, existing software tools for addressing them were evaluated.

Based on this evaluation, a software framework that enables developers of psychophysiologically interactive computer systems to address the associated challenges was developed. The framework supported the construction of modular software architectures by utilizing with the Pipes and Filters design pattern. The framework also enabled systems to adapt during their operation. This adaptability was supported by the use of software agent technology.

The framework was implemented in Java and C++ programming languages. Then, two systems were constructed with the framework in order to evaluate the support it offers for the construction of psychophysiologically interactive computer systems. The first system was a remote heart rate monitoring system. The second system was constructed for performing an empirical study involving both facial electromyographic and wireless electrocardiographic measurements. The first system was tested by four subjects who performed scenarios of voluntarily induced heart rate changes. Measurements from twenty-seven participants were collected with the second system. The results showed that the framework supported the construction of these systems and their accurate and reliable operation. The results also suggested that the framework supports extending these prototypes into robust real-world systems.

Acknowledgement

This research was supported by the Academy of Finland (project number 1202183).

References

[Allanson, 2002] Allanson, J. (2002). Electrophysiologically interactive computer systems. *IEEE Computer Journal*, 35 (3), 60-65.

[Allanson and Fairclough, 2004] Allanson, J. and Fairclough, S. H. (2004). A research agenda for physiological computing. *Interacting with Computers*, 16, 857-878.

[Arroyo and Childers, 1982] Arroyo, A. A. and Childers, D. G. (1982). A modular software real-time brain wave detection system. In *Proceedings of the 20th Annual Southeast Regional Conference,* 126-131.

[Bernardi *et al.*, 2000] Bernardi, L., Wdowczyk-Szulc, J., Valenti, C., Castoldi, S., Passino, C., Spadacini, G., and Sleight, P. (2000). Effects of controlled breathing, mental activity and mental stress with or without verbalization on heart rate variability. *Journal of the Americal College of Cardiology*, 35, 1462-1469.

[Binkley, 2003] Binkley, P. F. (2003). Predicting the potential of wearable technology. *IEEE Engineering in Medicine and Biology Magazine*, 22 (3), 23-27.

[Böcker *et al*., 1994] Böcker, K. B. E., van Avermaete, J. A. G., and van den Berg-Lenssen, M. M. C. (1994). The international 10-20 system revisited: cartesian and spherical co-ordinates. *Brain Topography*, 6, 231-235.

[Bodymedia, 2005] Bodymedia. (2005). Our Products in the Marketplace. Retrieved March 15, 2005, from http://www.bodymedia.com/consumer/overview.jsp.

[Bondmass *et al.*, 1999] Bondmass, M., Bolger, N., Castro, G., and Avitall, B. (1999). The effect of physiologic home monitoring and telemanagement on chronic heart failure outcomes. *The Internet Journal of Asthma, Allergy and Immunology*, 3 (2). Retrieved April 5th, 2005, from http://www.ispub.com/ostia/index.php?xmlFilePath=journals/ijanp/vol3n2/chf.xml.

[Brownley *et al.*, 2000] Brownley, K. A., Hurwitz, B. E., and Schneiderman, N. (2000). Cardiovascular psychophysiology. In Cacioppo, J. T., Tassinary, L. G., and Berntson, G. G. (Eds.). *The handbook of psychophysiology, 2nd ed.* (pp. 224-264). New York: Cambridge University Press.

[Buschmann *et al.*, 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., and Stal., M. (1996). *Pattern-Oriented Software Architecture*. Chichester, England: John Wiley & Sons.

[Busso *et al.*, 2004] Busso, C., Deng, Z., Yildirim, S., Bulut, M., Lee, C. M., Kazemzadeh, A., Lee, S., Neumann, U., Narayanan, S. (2004). Analysis of emotion recognition using facial expressions, speech and multimodal

information. In *Proceedings of the sixth international conference on Multimodal interfaces*, 205-211.

[Cacioppo *et al*., 2000] Cacioppo, J. T., Tassinary, L. G., and Berntson, G. G. (2000). Psychophysiological science. In Cacioppo, J. T., Tassinary, L. G., and Berntson, G. G. (Eds.). *The handbook of psychophysiology, 2nd ed.* (pp. 3-23). New York, USA: Cambridge University Press.

[Chao *et al*., 2004] Chao, D. L., Balthrop, J., and Forrest, S. (2004). Adaptive Radio: Achieving consensus using negative preferences. *Technical Report TR-CS-2004-08*. The University of New Mexico, Department of Computer Science. Albuquerque, USA. Retrieved March 24th, 2005, from http://www.cs.unm.edu/~dlchao/radio/.

[Chen and Vertegaal, 2004] Chen, D. and Vertegaal, R. (2004). Using mental load for managing interruptions in physiologically attentive user interfaces. In *Extended abstracts of the 2004 conference on Human factors and computing systems*, 1513-1516.

[Cohen, 2000] Cohen, A. (2000). Biomedical Signals: Origin and Dynamic Characteristics; Frequency-Domain Analysis. In Bronzino, J. D. (Ed.) *The Biomedical Engineering Handbook, 2nd ed*. Boca Raton, USA: CRC Press LLC.

[Cohen *et al.*, 1997] Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L. and Clow, J. (1997). QuickSet: multimodal interaction for distributed applications. In *Proceedings of the Fifth ACM International Multimedia Conference 1997*, 31-40.

[CollabNet, 2004] CollabNet, Inc. (2004). The Bean Builder: A BeanBox for the new Millennium. Retrieved April 5th, 2005, from https://bean-builder.dev.java.net/.

[Compumedics, 2005] Compumedics Ltd. (2005). Product Detail: Siesta. Retrieved March 23rd, 2005, from http://www.compumedics.com/product_detail.asp?id=13&item=product.

[Davies and Gellersen, 2002]Davies, N and Gellersen, H.-W. (2002). Beyond prototypes: challenges in deploying ubiquitous systems. *IEEE Pervasive Computing*, 1 (1), 26-35.

[Dey, 2001] Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, 5, 4-7.

[Elting *et al*., 2003] Elting, C., Rapp, S., Möhler, G., and Strube, M. (2003). Architecture and implementation of multimodal plug and play. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, 93-100.

[Felzer and Freisleben, 2000] Felzer, T. and Freisleben, B. (2000). HaWCoS: the "handsfree" wheelchair control system. In *Proceedings of the 5th International ACM Conference on Assistive Technologies*, 127-134.

[Fisher and Dourish, 2004] Fisher, D. and Dourish, P. (2004). Social and temporal structures in everyday collaboration. In *Proceedings of the 2004 conference on Human factors in computing systems*, 551-558.

[Fitzmaurice *et al*., 2003] Fitzmaurice, G. W., Khan, A., Buxton, W., Kurtenback, G., and Balakrishnan, R. (2003). Sentient data access via a diverse society of devices. *ACM Queue*, 1 (8), 53-62.

[Flippo *et al*., 2003] Flippo, F., Krebs, A., and Marsic, I. (2003). A framework for rapid development of multimodal interfaces. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, 109-116.

[Fridlund and Cacioppo, 1986] Fridlund, A. J. and Cacioppo, J. T. (1986). Guidelines for human electromyographic research. *Psychophysiology*, 23, 567-589.

[Gamma *et al.*, 1994] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Reading, USA: Addison-Wesley.

[Garcia-Palacios *et al.*, 2002] Garcia-Palacios, A., Hoffman, H. G., Carlin, C., Furness, T. A. III, Botella, C. (2002). Virtual reality in the treatment of spider phobia: A controlled study. *Behaviour Research and Therapy*, 40, 983-993.

[Gratton, 2000] Gratton, G. (2000). Biosignal processing. In Cacioppo, J. T., Tassinary, L. G., and Berntson, G. G. (Eds.). *The handbook of psychophysiology, 2nd ed.* (pp. 900-923). New York, USA: Cambridge University Press.

[Hinterberger *et al*., 2004] Hinterberger, T., Neumann, N., Pham, M., Kübler, A., Grether, A., Hofmayer, N., Wilhelm, B., Flor, H., and Birbaumer, N. (2004). A multimodal brain-based feedback and communication system. *Experimental Brain Research*, 154 (4), 521 -526.

[Hjortskov *et al.*, 2004] Hjortskov, N., Rissén, D., Blangsted, A. K., Fallentin, N. Lundberg, U., and Søgaard, K. (2004). The effect of mental stress on heart rate variability and blood pressure during computer work. *European Journal of Applied Physiology*, 92 (1-2), 84-89.

[Ijsselsteijn *et al*., 2004] Ijsselstein, W., de Kort, Y., Westerink, J., de Jager, M., and Bonants, R. (2004). Fun and sports: enhancing the home fitness experience. *Lecture Notes in Computer Science*, 3166, 46-56.

[Ilmonen and Kontkanen, 2003] Ilmonen T. and Kontkanen, J. (2003). Software architecture for multimodal user input - FLUID. *Lecture Notes in Computer Science*, 2615, 319-338.

[Johnson, 1997] Johnson, R. E. (1997). Frameworks = (Components + Patterns). *Communications of the ACM*, 40 (10), 39-42.

[Lisetti and LeRouge, 2004] Lisetti, C. and LeRouge, C. (2004). Affective computing and tele-home health. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, 148-155.

[Lisetti and Nasoz, 2002] Lisetti, C. And Nasoz, F. (2002). MAUI: a multimodal affective user interface. In *Proceedings of the tenth ACM international conference on Multimedia*, 161-170.

[Mainardi *et al*., 2000] Mainardi, L. T., Bianchi, A. M., and Cerutti, S. (2000). Digital biomedical signal acquisition and processing. In Bronzino, J. D. (ed.) *The Biomedical Engineering Handbook, 2nd ed*. Boca Raton, USA: CRC Press LLC.

[Marti and Lee, 2000] Marti, S. and Lee, K. H. (2000). The adaptive song selector or locator (ASSOL). Retrieved March 24th, 2005, from http://web.media.mit.edu/~stefanm/ass/ASSOL_20001219h_color3rd.pdf.

[Millán, 2003] Millán, J. del R. (2003). Adaptive Brain Interfaces. Communications of the ACM, 46 (3), 74-80.

[Moran *et al.*, 1998] Moran, D. B., Cheyer, A. J., Julia, L. E., Martin, D. L., and Park, S. (1998). Multimodal user interfaces in the Open Agent Architecture. *Knowledge-Based Systems*, 10, 295-303.

[Neuman, 2000a] Neuman, M. R. (2000). Biomedical sensors. In Bronzino, J. D. (ed.) *The Biomedical Engineering Handbook, 2nd ed*. Boca Raton, USA: CRC Press LLC.

[Neuman, 2000b] Neuman, M. R. (2000). Biopotential electrodes. In Bronzino, J. D. (ed.) *The Biomedical Engineering Handbook, 2nd ed*. Boca Raton, USA: CRC Press LLC.

[Nigay and Coutaz, 1993] Nigay, L. and Coutaz, J. (1993). A design space for multimodal systems: concurrent processing and data fusion. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 172-178.

[OMG, 2004] Object Management Group. (2004). UML™ Resource Page. Retrieved April 5th, 2005, from http://www.uml.org/.

[Oviatt and Cohen, 2000] Oviatt, S. and Cohen, P. (2000). Perceptual user interfaces: multimodal interfaces that process what comes naturally. *Communications of the ACM*, 43 (3), 45-53.

[Partala and Surakka, 2004] Partala, T. and Surakka, V. (2004). The effects of affective interventions in human-computer interaction. *Interacting with Computers*, 16, 295-309.

[Pentland, 2000] Pentland, A. (2000). Perceptual intelligence. *Communications of the ACM*, 43 (3), 35-44.

[PhysioNet, 2003] PhysioNet. (2003). PhysioToolkit. Retrieved April 5th, 2005, from http://www.physionet.org/physiotools/.

[Polar, 2004] Polar Electro. (2004). Fitness segment. Retrieved March 23rd, 2005, from http://www.polar-uk.com/html/segments/Fitness.html.

[Rangayyan, 2001] Rangayyanm R. M. (2001) *Biomedical Signal Analysis: A Case-Study Approach*. Wiley.

[Reithinger *et al*., 2003] Reithinger, N., Alexandersson, J., Becker, T., Blocher, A., Engel, R., Löckelt, M., Müller, J., Pfleger, N., Poller, P., Streit, M., and Tschernomas, V. (2003). SmartKom: adaptive and flexible multimodal access to multiple applications. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, 101-108.

[Russell and Norvig, 1995] Russell, S. and Peter, N. (1995). *Artificial Intelligence: A Modern Approach*. New Jersey, USA: Prentice Hall.

[Schmidt *et al*., 1996] Schmidt, D. C., Fayad, M., and Johnson, R. E. (1996). Software Patterns. *Communications of the ACM*, 39 (10), 37-39.

[Smart, 2004] Smart, J. (2004). wxWidgets: Cross-platform GUI library. Retrieved April 5th, 2005, from http://www.wxwidgets.org/.

[Strike and Steptoe, 2003] Strike, P. C. and Steptoe, A. (2003). Systematic review of mental stress-induced myocardial ischaemia. *European Heart Journal*, 24, 690-703.

[Sugar, 2005] Sugar, D. (2005). GNU Common C++ Resources. Retrieved January 12th, 2005, from http://www.gnu.org/software/commoncpp/.

[Sun, 2004] Sun Microsystems, Inc. (2004). Desktop Java: JavaBeans. Retrieved April 5th, 2005, from http://java.sun.com/products/javabeans/.

[Surakka *et al*., 2004] Surakka, V., Illi, M., and Isokoski, P. (2004). Gazing and frowning as a new human-computer interaction technique. *ACM Transactions on Applied Perception*, 1 (1), 40-56.

[Tassinary and Cacioppo, 2000] Tassinary, L. G. and Cacioppo, J. T. (2000). The skeletomotor system: Surface electromyography. In Cacioppo, J. T., Tassinary, L. G., and Berntson, G. G. (Eds.). *The handbook of psychophysiology, 2nd ed.* (pp. 163-199). New York, USA: Cambridge University Press.

[Teller, 2004] Teller, A. (2004). A platform for wearable physiological computing. *Interacting with Computers*, 16, 917-937.

[Tennenhouse, 2000] Tennenhouse, D. (2000). Proactive computing. *Communications of the ACM*, 43 (5), 43-50.

[Vehkaoja and Lekkala, 2004] Vehkaoja, A. and Lekkala, J. (2004). Wearable wireless biopotential measurement device. In *Proceedings of the Proactive Computing Workshop PROW 2004*, 29-31.

[VRMC, 2005] Virtual Reality Medical Center. Retrieved March 16, 2005, from http://www.vrphobia.com/index.htm.

[W3C, 2004] World Wide Web Consortium. (2004). Extensible Markup Language (XML). Retrieved April 5th, 2005, from http://www.org/XML/.

[Ward and Marsden, 2003] Ward, R. D. and Marsden, P. H. (2003). Physiological responses to different WEB page designs. *International Journal of Human-Computer Studies*, 59, 199-212.

[Wolpaw *et al*., 2002] Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., and Vaughan, T. M. (2002). Brain-computer interfaces for communication and control. *Clinical Neuropsychology*, 113, 767-791.

[Zeng *et al*., 2004] Zeng, Z., Tu, J., Zhang, T., Rizzolo, N., Zhang, Z., Huang, T. S., Roth, D., and Levinson, S. (2004). Bimodal HCI-related affect recognition. In *Proceedings of the sixth international conference on Multimodal interfaces*, 205-211.

[Zhai, 2003] Zhai, S. (2003). What's in the Eyes for Attentive Input. *Communications of the ACM*, 46 (3), 34-39.