A Middleware Architecture for Human-Centred Pervasive Adaptive Applications

Andreas Schroeder[1], Marjolein van der Zwaag[2], Moritz Hammer[1]

[1]*Institute for Informatics, Ludwig-Maximilians-Universität München*
Oettingenstr. 67, 80538 München, Germany

[2]*Philips Research*
High Tech Campus 34, 5656 AE Eindhoven, The Netherlands

[schroeda, hammer]@pst.ifi.lmu.de, marjolein.van.der.zwaag@philips.com

Abstract

As software is more and more interweaving with our everyday life, designing software in a way that it reflects and respects the user and her emotional physical conditions, cognitive engagement, and emotional state, become imperative. However, how such human-centred pervasive adaptive applications are to be designed and realized is still hardly understood. Both engineering approaches and runtime support for such applications are still in their infancy. In this paper, we present the REFLECTive middleware, a framework that facilitates the development and operation of such applications. The middleware is explained on the base of an envisioned example, the affective music player. By offering design patterns geared towards pervasive adaptive applications and leveraging them for achieving adaptivity, the REFLECTive middleware support a systematic and clear approach to engineering and deploying human-centred pervasive adaptive applications in daily life situations.

1. Introduction

Software and computer-aided systems are becoming more and more interwoven with our everyday life. Likewise, it is well understood that software systems have to adapt autonomously to the user an her context in order to increase their value to the user [18].However, the insight that the user context also consists of "non-functional aspects", i.e. her emotional state (such as anger), cognitive engagement (such as under challenged or overburdened) and physical conditions (such as temperature and fatigue), is relatively new [12].

How to design and realize such human-centred pervasive adaptive applications is still hardly understood, as so far most approaches to adaptive systems have been concentrating on "purely functional" aspects of the user.

Realizing human-centred pervasive adaptive applications is challenging in two ways: first, the user's requirements and intent cannot be queried directly, but need to be inferred from unobtrusive, non-disruptive measurements of the user and her environment. Second, foreseeing possible adaptation dimensions of a pervasive application and realizing them in the software is a challenging task requiring a careful and systematic design approach.

In this paper, we therefore propose a middleware, the REFLECTive middleware, that is intended to leverage existing insights in reconfiguration of component-based software and adaptive applications [13] for the development of human-centered pervasive adaptive applications.

Together with the middleware, we present a set of design patterns geared towards pervasive applications, and discuss the benefits they offer for designing extensible applications which are adaptable to the user's state and her context.

This paper is structured as follows. We discuss the particularities of our target application domain and of an example application, the affective music player, in Section 2. We present the REFLECTive middleware, its principles and patterns for the design of human-centred pervasive adaptive applications in Section 3. Section 4 shows how the affective music player is designed using the REFLECTive middleware. We discuss related work in Section 5, and conclude in Section 6.

2. Human-Centred Pervasive Adaptive Applications

Pervasive adaptive applications are applications that adapt to the situation and state of the user automatically and interact with the user in a pervasive way. That is to say, the application seamlessly integrates in the environment of the user, without an explicit computer-flavoured interface such as mouse, keyboard, and screen [18] [4].

Within REFLECT, we utilize user-centred research in order to assure that the adaptive application makes perfect sense to the user. Accordingly, instead of considering only the physical and IT environment of a user, the user's cognitive (e.g. workload), affective (e.g. mood), and physical (e.g. movements) conditions are

also taken into account. The purpose of adaptation in such a setting is twofold: first, applications reactively follow the user's state, and second, applications proactively seek to improve the user's state. A pervasive adaptive application as considered in this paper is hence an application which acts as a closed-loop system: the environment and user's state are measured, actuators adapt to and influences this state, which is again measured and thereby feeds back into the adaptation mechanism.

To investigate and illustrate pervasive adaptive systems, we consider two application domains: automotive and desk work environments. These scenarios are beneficial since they share interactive environments in which we can constrain ourselves to focus on a single seated user while retaining the possibility to operate sensors, gather measures that determine the current state of the user, and test requirements for adaptation actuators. Note that the single-user simplification is necessary since the field is still in an explorative phase, and the pervasive adaptive systems have to deal with a large amount of requirements making them highly complex.

Pervasive adaptive systems often have to handle both data streams (e.g. video, audio, and psychophysiologial sensor signals) and external events (e.g. the beginning or ending of a video signal). These requirements are further described on the basis of an envisioned pervasive adaptive system: the affective music player.

The affective music player functions as a closed loop repeatedly measuring the current mood state of a user and selecting music from the user's own music database depending on the current mood and a predetermined target mood state. Since a positive mood enhances several cognitive processes, the ability to improve mood is particularly interesting (e.g. for car driving or during work) [7]. Note that mood is seen as a long lasting (i.e. minutes to days) affective state with no clear cause of origin [17]. Moods are thus different from emotions, which are short lasting (i.e. seconds to minutes) affective processes related to an object (i.e. an event in the environment or inner thought) [1]. In the affective music player the target mood state is selected by the user itself to increase the acceptance of the system [10] [19].

Unobtrusive mood measures follow from the neurological underpinning of moods which show that moods are reflected, and thus can be measured via skeletal- muscular system activity (i.e. image processing or face expression). For example, the muscles at the corner of the mouth (i.e. the zygomaticus major muscles) are more contracted when a person is happy compared to sad [7] .. Furthermore, information to indicate the current mood can be gathered from autonomous nervous system activity (i.e. psychophysiological measures as cardiac activity and skin conductance) [17]. The use of multiple measures is needed to be sensitive enough to measure all dimensions of mood, as well as to increase robustness. Before mood states can be interpreted from the signals in real time, several preprocessing steps have to be taken into account. For example, determining the face expression in a certain time interval requires real time image processing, measure extraction, and classification of the measures. Psychophysiological signals need even more preprocessing steps ranging from filtering and interpolation from movement artifacts, to the acquisition of the desired measures. Furthermore, it must be noted that the values of the unobtrusive mood measures vary among individuals. For this purpose it is important for a human-centered pervasive adaptive application (in the following also called REFLECTive applications) to have the ability to learn the individual patterns of each user to optimize its performance.

The selection of the right song implies further requirements. Besides the fact that the effects of music on mood are personal (i.e. everyone has its own music taste and memories for particular music), each song influences the user's mood in a specific direction and this direction is dependent on the current mood of the user. Thus the system must have the ability to make predictions of the effects of a song on mood, based on previous expositions to that song or measures from users with a comparable profile.

Of course, different actuators besides music can be used to adapt the environment (e.g. light). These actuators differ in their timing properties, i.e. how fast, accurate, and with what ranges they can be adjusted (e.g. disrupting songs is to be avoided, so that music can be altered about every three minutes; light can be adjusted at every desired moment). Input and output options of REFLECTive systems are thus diverse and each has its own properties. To handle this complexity in a practical way a good architecture is needed.

Altogether it can be summarized that developing human-centred pervasive adaptive applications is a challenging task due to the high complexity of adaptation logic, the uncertainty involved in the interpretation of measures, and diversity of involved technologies. In order to alleviate this task, we propose to follow a middleware approach in which the middleware offers functionality and architectural principles allowing reu of the existing system functionality and symplifyingthe development of pervasive adaptive applications.

3. A Middleware for Pervasive Adaptive Applications and its Principles

In the REFLECT project, we develop a middleware reducing the development effort necessary to realize REFLECTive applications. On the one hand, the

REFLECT middleware provides functionality that is commonly needed in REFLECTive applications and therefore prevents the software engineer from reinventing the "REFLECTive wheel". On the other hand, the REFLECT middleware features an architecture that facilitates the decomposition of pervasive adaptive applications into manageable implementation tasks as well as the extension of applications with new features at run-time.

More specifically, the middleware is based on the principles of Component Based Software Engineering (CBSE [16]) and closed control loop structures, like e.g. in the affective music player. By following these principles, the middleware allows to decompose the complex task of self-adapting software into small control loops that deal with self-adaptation issues on a level of abstraction that allows for compact and clutter-free realizations.

3.1. Component Based Software Engineering

CBSE is a paradigm focussing on the decomposition of systems into *components*, communicating with each other and their environment through *ports*, which in turn are connected by *connectors*. Connectors between ports may be established only if the *port types*, defining the offered and required functionality as well as the behaviour, are compatible. While the exact definition of compatibility differs between component models, a canonical basis of compatibility is established by requiring that all functionality required by one port must be offered by the other port, and vice versa.

Using CBSE to realize a software/hardware system brings several advantages: it allows decomposing the realization task into subtasks, thereby separating independent concerns, encourages defining clear interfaces also within the realized system, and allows for independent implementation and maintenance.

The decomposition approach of CBSE is often also applied to components themselves, decomposing components into smaller, finer-grained units where necessary.

Using CBSE also promise a high level of reusability. Once functionality is encapsulated within a component and offered through one of its ports, the functionality can be reused by other components. This aspect is highly attractive for human-centred pervasive adaptive applications, as makes the system flexible and easily extensible – a necessity in the anticipation of new insights in the highly innovative and fast-changing application domain. It also allows pervasive adaptive applications being built using off-the-shelf components or even conjointly by several specialized manufacturers, leveraging the know-how of each manufacturer to create sophisticated applications.

Additionally, a component-based approach facilitates making the system modifiable at run-time, by creating or removing components, or altering connections between components.

3.2. Content-Specific Connectors

In human-centred pervasive adaptive applications, a variety of data is transported between components, ranging from binary media or sensor data streams to discrete control messages. In order to describe these different connections in a uniform manner, we augment the role of the connectors. Depending on the nature of the data that needs to be transported, a connector can be an active, stateful entity that transports data between ports – e.g., by reading from a video buffer and streaming the data to a remote target, or by queuing control messages sent and delivering them to the target port.

The benefits from such an approach are twofold: First, the component implementation only needs to provide data to the (local) connector endpoint instead of choosing adequate transportation means to a (possibly remote) target port. Second, during reconfiguration, the connector can handle its own state (e.g., the messages received but not yet delivered, or a sensor data buffer) and ensure that no vital data is lost. To this end, connectors need to provide a number of reconfiguration primitives, like their endpoint becoming detached from one component and attached to another.

3.3. Decomposing Adaptation Logic

An application that influences the physical environment of a user and monitors the results achieved can be seen as a closed-loop system. This understanding – stemming from control theory – provides a helpful mental model for developing adaptive systems. However, a full control-theoretical approach only excels at handling closed systems with properties known at design time and which can be modelled using specific linear or nonlinear mathematical equations. Therefore, a strict control-theoretical approach is too limited for an adaptive system which must be able to run in open environments. Nevertheless, the control loop approach is very influential in the software engineering domain as a general organization principle for software systems [14], as it conceptualizes a clear separation of adaptation tasks from computation tasks.

The adaptation task itself can be split into four phases, Monitoring, Analysis, Planning, and Execution [9] (MAPE, cf. Figure 1), which are described in the following. The monitoring phase handles the gathering of data by means of probes installed in the system. Analysis is concerned with aggregating and interpreting the raw data gathered, and thereby producing information about the user and the context on a higher level of abstraction. The third step, "Planning", deals

with inferring the actions to take based on the available data and goals, and initiate these actions (note that although the name refers to a technique of artificial intelligence, the planning phase does not necessarily have to be realized by a planner). The fourth and last phase, "Execution", finally deals with the actual execution of the inferred actions.

The breakdown of the adaptation loop into four phases is only one of two possible decomposition dimensions. Adaptation logic may also be decomposed into a hierarchy of control loops, where the higher level control loops control the lower level loops (cf. Figure 1). Decomposing the adaptation logic not only into different phases, but also into hierarchies allows handling adaptation issues on the appropriate level of abstraction. The affective music player for example, while directing or following the mood of a user, also needs to recognize the circumstances under which it should be operating and those under which it should not. This concern may be handled in the next higher level of adaptation logic, taking more context information and information about the user into account than is needed for the sole affective music player, and thereby allowing the affective music player to solely focus on improving the user's mood by selecting an appropriate piece of music.

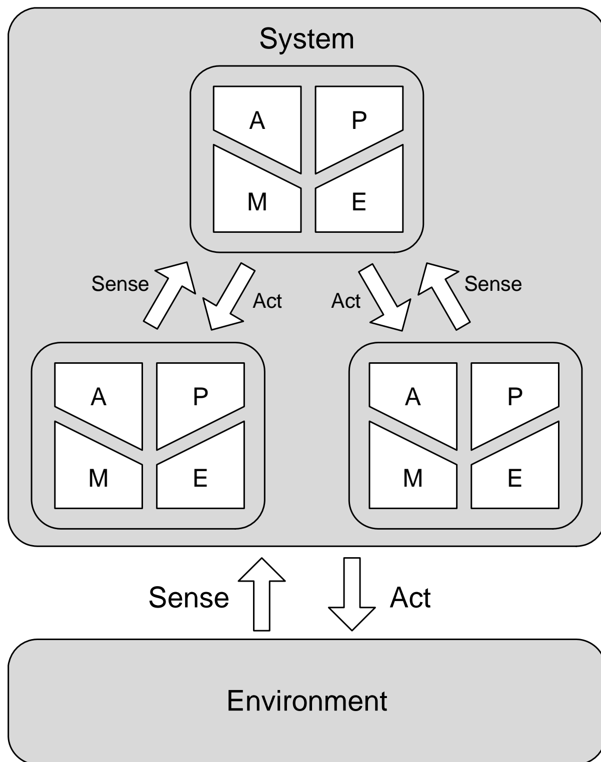

Figure 1. Separating the Control Logic

Note that in the REFLECTive middleware, we suggest using hierarchical control structures instead of peer-to-peer adaptation mechanisms; this is for reasons of conceptual simplicity only. As approaches exist on how to distribute a centralized control algorithm over equal peers, distributed peer-to-peer control can be understood as a (fairly non-trivial) refinement of a centralised control approach. This complication in design and technique is left aside for the first stage of the REFLECTive middleware.

3.4. Achieving Flexibility

Flexibility is a hard problem for software design, and infeasible to achieve without the help of a middleware that provides distinctive means. The REFLECTive middleware provides three key features to support adaptivity.

Firstly, adaptation usually employs a sequence of fine-grained, technical steps. The REFLECT middleware allows to define a domain specific language (DSL) over the reconfiguration primitives it provides, thereby allowing to abstract from the technical approach and describing reconfigurations on a higher level. The usefulness of this approach is amplified by the hierarchization of control loops and components, as a DSL can give a concise description of a high-level adaptation that translates to a extensive schedule on the implementation level.

Secondly, the middleware can offer additional adaptation dimensions by offering a flexible instantiation based on architectural styles [6]. An architectural style represents – from the CBSE perspective – a set of constraints on the system architecture defining the allowed and required components and their interconnections. From an application point of view, an architectural style represents an application profile in that it describes a set of mandatory components for the realization of an application. This application profile can be used to instantiate an adhering subsystem, and thereby dynamically creating and launching the application at run-time using components known by the middleware.

Third, the REFLECTive middleware stores a model of its application's structure at runtime, allowing analyzer and planner components to reflect on the architectural style used. Storing this data explicitly is necessary to retain the hierarchical information and architectural style descriptions that cannot be inferred from the system by runtime reflection. This model can also be used for predicting the system's behaviour for the future, which is important for pre-emptive adaptation. It is also required for verifying the validity of a reconfiguration description before its execution.

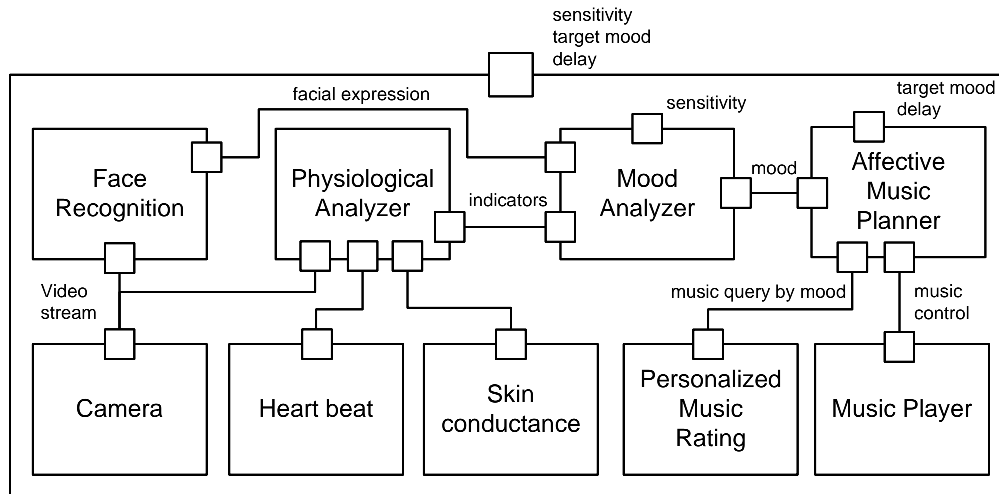

Figure 2. The Affective Music Player

4. Designing the Affective Music Player

Using the decomposition approach proposed in section 3.3, the affective music player can be designed as depicted in Figure 2. The architecture presented has a clear separation of tasks that need to be addressed in order to realize the overall system. First of all, data about the user must be gathered through a camera, heart beat and skin conductance sensor. The camera data is analyzed in a face recognition component, with the result being fed into a mood analyzer. Skin conductance and heart rate is analyzed in the physiological analyzer which also takes into account information from the camera to predict measure disturbances due to motion. The mood analyzer determines, based on the recognized facial expression and physiological information, the current mood state of the user. The affective music planner is connected to the mood analyzer and therefore receives data about the current mood of the user. If the music planner sees fit, it can query a personalized music rating service providing a set of music pieces that might improve the user's mood towards desired target mood. It then arranges the songs into a playlist, and uses the music player to process its playlist, continuously monitoring the achieved changes in the user's mood.

Adopting this architecture and realizing the system using the component based approach as illustrated eases foreseeing possible adaptations of the system. For example, the architecture shows that psychophysiological information may also be derived from other measures. At the same time, following the hierarchical control loop paradigm encourages developers to think about configuration parameters that may be exposed to higher level controllers. In the example, the configuration parameters exposed may be the target mood – which may be controlled through a user interface or another adaptive system –, the delay between sensing a mood change and reacting, and the sensitivity of the mood analyzer.

5. Related Work

A number of component frameworks supporting adaptation have been developed, for overviews see [11], [1]. These frameworks differ in the granularity of components as well as their provisions for triggering and planning the reconfiguration.

Similar to our approach are Rainbow [5] and KX [8], which use probes and gauges to generate and aggregate streams of measurements for a running application. A controller then makes reconfiguration decisions, which are actuated by the gauges' output. The probes, however, are software entities used to provide measurement from legacy systems ([8]) or generic components ([5]), whereas, in our system, they represent physical sensors.

CADDMAS [15] is a framework for adaptive processing of signals from sensors, i.e. streams of data. Because of terse real-time constraints, a finite set of alternative configurations is precalculated and adaptively chosen from at runtime. In the absence of hard real-time constraints, we feel that this approach is too limiting in a generic setup.

CINEMA [1] and Djinn [10] are frameworks that operate on media data streams. Both emphasize the use of streams to transport data and handle discrete control events on a different level. Adaptation is triggered by explicit user intervention. The major focus is placed on "smoothness"-preserving reconfiguration, i.e. reconfiguration under real-time constraints.

6. Conclusion

As pervasive adaptive applications get more and more attention from industry and research, systematic

software engineering approaches become imperative. In this paper, we have motivated that component based approaches yield several benefits for the design and the run-time flexibility of pervasive adaptive applications.

Our main contribution to the engineering of pervasive adaptive applications is twofold. First, we motivate that component based software engineering (CBSE) eases the systematic design of application and adaptation logic, and presented patterns for decomposing adaptation logic into four phases and hierarchically, achieving a clear separation of concerns. Second, we presented three aspects in which CBSE can be further leveraged for achieving flexible and autonomously adaptive systems: using the reconfiguration primitives of components and connectors to create domain specific languages for adaptation, using architectural designs as application profiles to flexibly instantiate applications at runtime, and predicting the system behaviour based on design-time models.

We believe that leveraging component based software engineering principles as described is a key factor for the systematic and successful development of human-centred pervasive adaptive applications, which more and more interfuse our everyday environment.

We are currently implementing the REFLECTive middleware following the principles outlined in this paper in order to test our hypothesis, and will report on the outcome in future work.

Acknowledgements

This work has been partially sponsored by the EC project REFLECT, IST-2007-215893.

References

[1] I. Barth, Configuring Distributed Multimedia Applications using CINEMA. International Workshop on Multimedia Software Development (MMSD '96), 1996

[2] J.S. Bradbury, J. Cordy, J. Dingel and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. WOSS '04. ACM. 28-33. 2004

[3] A. D. Damasio. Descartes' error: Emotions, reason, and the human brain. G.P. Putman, New York. 1994.

[4] Z. Duric, W. D. Gray, R. Heishman, F. Li, A. Rosenfeld, M. J. Schoelles, C. D. Schunn, and H. Wechsler. Integrating perceptual and cognitive modeling for adaptive and intelligent human-computer interaction. *Proc. IEEE, 90*(7), 1272-1289. 2002.

[5] D. Garlan, S. Cheng, A. Huang, B. Schmerl and P. Steenkiste. Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure. IEEE Computer, 37. 2004

[6] D. Garlan and M. Shaw. An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering. World Scientific Publishing Company, 1-39. 1993

[7] A. M. Isen. Positive Affect and Decision making. *In: M. Lewis, J. M. Haviland-Jones, and L. Feldman Barrett (eds.). Handbook of Emotions.* The Guilford Press, New York. 2000.

[8] G. Kaiser, P. Gross, G. Kc, J. Parekh and G. Valetto. An Approach to Autonomizing Legacy Systems. Workshop on Self-Healing, Adaptive and Self-Managed Systems. 2002

[9] J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Comput. 36(1)*, 41–50. 2003.

[10] S. Mitchell, H. Naguib, G. Coulouris and T. Kindberg. Dynamically reconfiguring multimedia components: a model-based approach. Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications. ACM, 40-47. 1998

[11] D. A. Norman. The design of future things. Basic Books, New York. 2007

[12] R. W. Picard, J. Klein. Computers that recognise and respond to user emotion: theoretical and practical implications. *Interacting with Computers,* 14(2), 141-169. 2002.

[13] S. M. Sadjadi, P. K. McKinley. A Survey of Adaptive Middleware. Technical Report, Computer Science and Engineering, Michigan State University. 2003

[14] M. Shaw. Comparing Architectural Design Styles. IEEE Software 12(6), 27-41. 1995.

[15] J. Sztipanovits, G. Karsai and T. Bapty. Self-adaptive software for signal processing Commun. ACM, 41. 66-73. 1998.

[16] C. Szyperski, D. Gruntz and S. Murer. Component Software: Beyond Object-Oriented Programming. ACM Press and Addison-Wesley. 2002

[17] R. E. Thayer. The biopsychology of mood and arousal. Oxford University Press, New York. 1989.

[18] M. Weiser. The computer for the 21st century. *Scientific American,* 265 (3), ,94-104. 1991

[19] P Wilhelm and D. Schoebi. Assessing Mood in Daily Life. *European Journal of Psychological Assessment,* 23(4). 258- 267. 2007