A Component-Based Approach for Realizing User-Centric Adaptive Systems

Gilbert Beyer, Moritz Hammer, Christian Kroiss, and Andreas Schroeder

Ludwig-Maximilians-Universität München
{beyer,hammer,kroiss,schroeda}@pst.ifi.lmu.de

Abstract. We discuss a generic architecture for building user-centric systems. The characteristic feature of such systems is a control loop that monitors the user's state, and produces a harmonized response. In order to adaptively respond to changes of the user's state, we propose an architecture with supervising loops. This allows the primary control loop to be written in a straight-forward way, and add adaptivity on a different level. We illustrate our approach with an example scenario that describes computer vision based adaptive interaction.

Keywords: Components, reflective systems, adaptation, reconfiguration.

1 Introduction

Software and software-enhanced systems are becoming more and more interwoven with our everyday life. Likewise, it is well understood that software systems have to pervade and adapt autonomously to the user and the environment in order to increase their value to him [Mar99]. The idea that software systems also consider the "nonfunctional aspects" of the environment and especially the user, i.e. the emotional state (such as anger or joy), cognitive engagement (such as high attention or distraction) and physical conditions (such as body posture and fatigue), is however relatively new [PK01].

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 delivering "pure functionality" to the user, e.g. providing enhanced information through location-based services.

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

In this paper, we present the vision and concepts of the REFLECTive framework and middleware: a software framework that is intended to facilitate the construction of user-centric pervasive adaptive systems. In the next Section, we first clarify our vision on user-centric pervasive adaptation. Then, we present our

software-engineering approach to the design of user-centric pervasive-adaptive software, allowing to create well-structured and flexible software applications in Section 3, and discuss their benefits in one example scenario in Section 4. Thereafter, we discuss related work in Section 5, and conclude in Section 6.

2   User-Centric Pervasive Adaptation

User-centric pervasive adaptive systems try to improve the well-being of a user by influencing environmental and workload-related parameters with the goal of improving the user's physical, emotional, and cognitive state. In order to infer the user's state, a user-centric pervasive adaptive system use available context information (e.g. is the user currently driving for hours in a traffic jam), and psycho-physiological measures (e.g. skin conductance, heart rate, or EEG). This data allows to fine-tune the environment of the user to create a more comfortable setting.

However, realising such systems brings its own challenges. For example, it is never guaranteed that an action performed on the environment (such as e.g. changing the lighting conditions) will show the desired effects, or any effects at all: preferences vary greatly over persons and over time. User-centric pervasive adaptive systems will therefore have to self-asses their effects on their environment, and hold several alternative strategies ready to achieve their goal. Furthermore, extracting the user's conditions from low-level psycho-physiological measures or computing it through image processing is a challenging task with varying success depending on the user's context; detecting the facial expression under bad lighting conditions for example is very difficult, if not impossible at all. These challenges call for a well-structured software solution that allows for flexible response to the users environment, and self-assessment of its performance. We foresee that in order to tackle this application domain, the software system built adapting the users environment and thereby guiding the users conditions towards well being needs to be adaptive and flexible in itself.

As the ability for internal restructuring to our eyes is one key enabler for building functioning feedback loops, one of the challenges in our approach to user-centric pervasive adaptive systems therefore is how to enable and facilitate this internal adaptation; This question is answered in the following sections.

3   Realizing User-Centric Software

Our primary approach towards user-centering are control loops. Realizing such loops, however, is a challenging task of software engineering. Given the unpredictability and uncertainty involved when considering a human user, control loops need to be flexible and adaptive, possibly need to integrating multiple sensor readings and are subjected to long-term changes of the application's goals. In order to facilitate the development of such loops, we propose an architecture that tries to address the various concerns involved in control loops separately.

For a clean structure of control loops, we suggest to distinguish a number of phases that are executed in sequence within the loop. We adopt the MAPE

scheme as proposed by IBM [KC03]. This scheme separates four distinct stages: Monitoring, Analysis, Planning and Execution. First, data needs to be obtained by monitoring. This data is analysed in the next stage, resulting in a concise description of the state. This description serves as input to the third stage, which is concerned with planning an appropriate action. Finally, the fourth stage executes the plan.

By separating the stages, a clear view on the various aspects to be considered when devising a control loop is obtained. In the context of user-centric systems, each stage poses its own challenges: monitoring needs to access physical sensors, possibly preparing the data, analysis needs to provide an abstract description of the user's state. Planning then needs to produce a plan that integrates the application's goals (e.g., mood improvement) and the observed user's state. This plan is to be executed in the last stage, using available actuators like displays or sound devices. Each of these stages is difficult to realize even without considering adaptivity. On the other hand, each stage is more or less required to be adaptive: the monitoring phase might have to address the loss of physical sensors, while the planning phase might need to respond to the failure of previous plans to achieve the intended effect.

In order to keep the realization of control loops tractable, we suggest to externalize the concern of adaptivity. Instead of putting adaptivity-related code into the MAPE loop stages, we suggest to supervise the MAPE loop, and replace some of its stages, should they prove inadequate. This externalization is obtained by two design elements: the use of *components* to build the control loops, and the use of *hierarchical* loops to obtain supervision.

Components have been among the first concepts provided for obtaining a modular design of software. Basically, they are black-box entities of software that explicitly declare their communication endpoints. In our approach, components declare a number of *ports*, which declare *provided* and *required* communication capabilities. Ports of different components can be connected by means of *connectors*. Since we need to handle a variety of communication styles (e.g., sensor readings which are transported from hardware layers to the monitoring components, or discrete messages that communicate the abstracted user state from the analysis stage to the planning components), we suggest to use *fat connectors*, which handle the implementation of the communication (as opposed to predefined communication means supported by the framework).

In order to facilitate clear application design, we also propose the use of hierarchical components. A component can then either be an atomic component, implemented in a host programming language, or a composite component that consists of a number of components and connectors that connect some of the ports of these components. The ports that are not connected within the hierarchical components become the ports the hierarchical components exports. Eventually, an entire application can be built as a hierarchical components that does not export any ports.

By building applications (or, for our purpose, stages of a MAPE loop) by selecting and connecting components, the architecture remains visible. It can

then be changed at runtime, in a process called *reconfiguration*. Reconfiguration can replace components, add or remove filter components to connections, or just modify a component's parameters. We suggest to realize adaptivity in this way: instead of integrating different strategies within a single component, we propose that reconfiguration is used to obtain adaptivity. For example, if a sensor fails, the monitoring component (which should be a composite component, consisting of various components that obtain and process the data) should be reconfigured to continue without the sensor.

Doing reconfiguration is not trivial, however. For utilizing reconfiguration as a means to obtain adaptivity, the four stages of the MAPE loop need again to be implemented: first, the application that might become reconfigured needs to be observed. From this observations, an abstract state – and a decision about the necessity of reconfiguration – needs to be generated. If reconfiguration is deemed necessary, it has to be planned in order to obtain an architecture that is more suitable. Finally, the plan has to be executed in a safe way (i.e., the reconfiguration must not disrupt ongoing communication [KM90]).

We hence propose a higher-level MAPE loop that supervises the low-level control loop and changes it by the means of reconfiguration if the low-level control loop is not performing as intended. Obviously, the higher-level MAPE loop can again be subjected to supervision, achieving a way to change long-term goals of an application.

4   Example Scenario: Adaptive Interactive Installations

A possible field of application of our proposed architecture for adaptive systems can be found in interactive installations in public spaces. Examples of such user-centric media are interactive tables, displays or projections in museums, exhibitions or other public buildings, providing topic-oriented content when users interact with them. Interactive installations can realize both a 1-to-1 dialogue with the user as well as interaction with a group of visitors [JS04]. 1-to-1 dialogue means that the installation communicates with a single user or is explicitly controlled by him. Installations that communicate to many people are put into effect by multi-user environments that offer a shared experience. The requirements for both types of user interaction are typically different. For example it often doesn't make sense to let an interactive object or virtual world be controlled by more than one user the same time; an interactive game instead may require a minimum of participants to get evident. The application designer has to decide to comply with one scenario that best fits the presumed situation. Interactive installations that are able to adapt to variable user situations (cf. Fig. 1a and 1b) are an example of applications that could expediently make use of the component-based approach and hierarchic MAPE loops.

The hierarchy of MAPE loops in this scenario can be structured as follows (cf. Fig. 2): an inner loop would realize the preferred interaction (e.g. explicit control of content by a camera sensor and hand gestures) and update the displayed content accordingly. A higher-level MAPE loop could assess if the difficulty level

(a) 1-to-1 Dialogue   **(b)** Multi-user Game
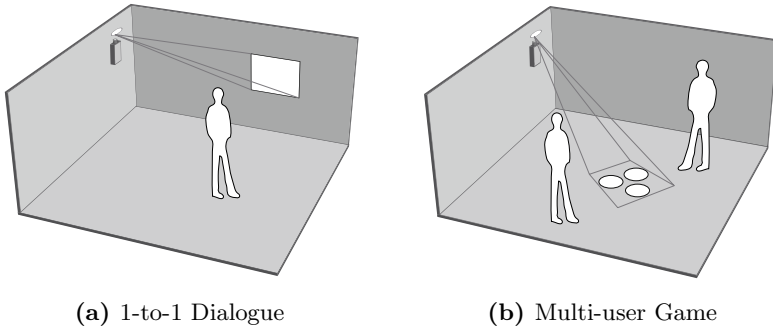
Fig. 1. Adaptive Interactive Installation

for the user has to be adjusted (e.g. if the user is moving his hands too slow) and adapt the corresponding parameters of the underlying interaction control loop. At the third level, another MAPE loop could monitor the current situation of the system (e.g. the number of users within range). In this step, the system would assess if the requirements for the chosen interaction are still fullfilled or if the content type of the lowest MAPE loop has to be changed, e.g. from one that supports one-person to one that supports multi-user interaction. To realize an adaptation like this by parameter adjustment can be quite cumbersome. Instead, the involved component within the lowest MAPE loop could be replaced by others that realize the desired interaction control.

As a way to facilitate the creation of such pervasive adaptive systems, the Java-based REFLECT component framework [WBH+09] is developed. It is built
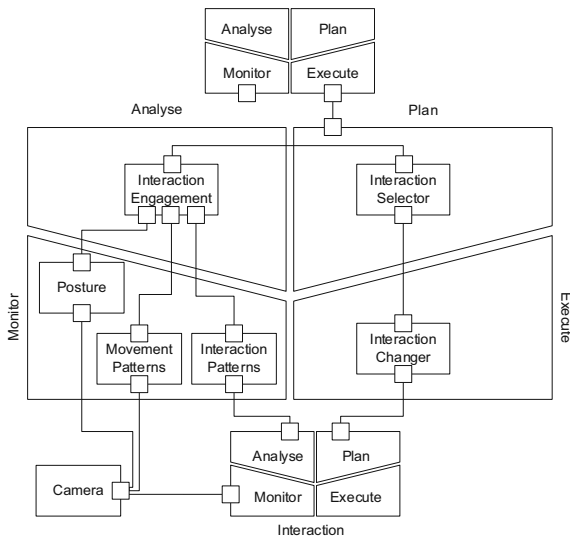
Fig. 2. Scenario Structure

on top of the OSGI service platform [OSG05] and provides a hierarchical component model with means for reconfiguration and reflection. Based on this foundation, some experimental applications were realized to evaluate the applicability of our approach and to test the framework. These applications instantiate the general scenario described above and focus on its technological challenges. The component-based approach and the concept of hierarchical and reconfigurable MAPE loops could be veryfied to be highly beneficial, especially as the involved computer vision techniques require the combined use of several complex algorithms. These could be very well structured as interlinked components that are adaptable by parameter-adjustment or reconfiguration. Additionally, rapid prototyping and experimentation were highly facilitated by assembling applications from reusable components. This is very important since user-centric adaptation constitutes a relatively new field of research.

5   Related Work

A large number of middlewares supporting adaptivity have been developed. A rather broad overview is given by Sadjadi et al. [MSKC04]. Huebscher and McCann [HM08] focus on the utilization of control loops for adaptivity, identifying an extended MAPE loop as a common denominator.

In [KM07], a three-layered architectural reference model for self-managed software is described that comprises layers for component control, change management and goal management, respectively. Reconfiguration is regarded as the essential adaptation mechanism of the component control layer. The change management layer contains a collection of change plans that are executed as reactions to state changes. The goal management layer is concerned with producing new change plans according to high-level goals. Altogether, this architecture could be interpreted as a hierarchy of MAPE loops with distinct responsibilities.

6   Conclusion and Future Work

We have presented an approach towards engineering user-centric systems. Since user-centric systems need to be constantly aware of the user's state, our approach builds on control loops. Implementing such loops is not easy, and we propose a two-fold separation of concerns for facilitating the development: considering discrete stages, and externalizing the adaptivity to a supervising loop. As part of the REFLECT project we have built a framework that supports this way of building user-sentric applications, and gathered first experience with interactive display systems.

For building larger applications, we consider further support for the MAPE loops to be necessary. This especially holds true for the supervising MAPE loops: analysing the need for a reconfiguration requires detailed knowledge about the purpose of the lower-level MAPE loop, and planning a reconfiguration needs additional knowledge about the alternative architectures. We believe that future work should address this issues, since automating parts of the MAPE loop appears a necessity for large-scale applications.

References

[HM08]   Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Computing Surveys 40(3), 1–28 (2008)

[JS04]   Sauter, J.: Das vierte format: Die fassade als mediale haut der architektur. In: Digitale Transformationen. Medienkunst als Schnittstelle von Kunst, Wissenschaft, Wirtschaft und Gesellschaft. whois verlags und vertriebsgesellschaft (2004)

[KC03]   Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)

[KM90]   Kramer, J., Magee, J.: The evolving philosophers problem: Dynamic change management. IEEE Transactions on Software Engineering 16(11), 1293–1306 (1990)

[KM07]   Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: Future of Software Engineering (FOSE 2007), pp. 259–268. IEEE Computer Society Press, Los Alamitos (2007)

[Mar99]   Mark, W.: The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev. 3(3), 3–11 (1999)

[MSKC04] McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: A taxonomy of compositional adaptation. Technical Report MSU-CSE-04-17, Department of Computer Science, Michigan State University (2004)

[OSG05]   The OSGi Alliance. OSGi service platform core specification release 4 (2005), `http://www.osgi.org/Release4`

[PK01]   Picard, R.W., Klein, J.: Computers that recognise and respond to user emotion: theoretical and practical implications. Interacting with Computers 14(2), 141–169 (2001)

[WBH⁺09] Wirsing, M., Beyer, G., Hammer, M., Kroiss, C., Schroeder, A.: RE-FLECT - first year report: Requirements and design. Technical report, LMU München (2009)