A Contract-Based Approach to Adaptivity in User-Centric Pervasive Applications*

Martin Wirsing, Moritz Hammer, Andreas Schroeder, and Sebastian Bauer

Ludwig-Maximilians-Universität München, Germany
{wirsing, hammer, schroeda, bauerse}@pst.ifi.lmu.de

Abstract. Pervasive user-centric applications operate in highly dynamic and uncertain environments. Designing such applications as one monolithic component taking all possible environments into account inevitably leads to bad system design. We instead propose constructing partial solutions handling only a subset of all possible environments, and changing the system as the environment evolves. We use an assume-guarantee contract framework to infer the conditions under which configurations exhibit the desired functionality. Furthermore, we show how a system undergoing reconfigurations can be shown to satisfy a global assume-guarantee contract.

1 Introduction

Pervasive user-centric applications are applications running on systems seamlessly integrating with their environment and adapting it to the user's current emotional, cognitive, and physical state [6]. Such applications must deal with several challenges:

- They must operate in highly dynamic and uncertain environments.
- They often interact with other agents and humans.
- They must remain non-disruptive, and at the same time improve the experience of the user.

In this paper, we take a closer look on how the first challenge in the development of user-centric applications can be handled: dealing with highly dynamic and uncertain environments. Our approach involves making explicit assumptions about the environment and the functionality of the system using assume-guarantee contracts[3]. We annotate the components constituting the system with assume-guarantee contracts in order to verify whether a global contract can be satisfied. If this is not the case, we introduce components monitoring the system. By reconfiguring the system as soon as one of the monitored conditions get falsified, we achieve adaptivity within the bounds of a given specification defining the desired system behaviour. Furthermore, knowing the assume-guarantee pairs of components, the reconfiguration rules and the overall contract to satisfy, we can verify whether the system under construction satisfies the global system contracts that describe its overall purpose.

* This work has been partially supported by the EC project REFLECT, IST-2007-215893 and the GLOWA-Danube project 01LW0602A2 sponsored by the German Federal Ministry of Education and Research.

The remainder of this paper is structured as follows. In Section 2, we give an informal overview of our contract-based approach to adaptivity. Following this informal description, Section 3 gives an example scenario for the usage of our approach, before we introduce the formal assume-guarantee framework in Section 4. We conclude in Section 5.

2 Contract-Based Approach to Adaptivity

User-centric pervasive adaptive systems need to be able to cope with a large amount of uncertainty. Sensor readings might not be available or be inaccurate, and actuators might fail to achieve the desired effect on the user. Different users might respond differently to stimuli supplied by the system, and users might become bored by a stimulus after different numbers of repetitions. In order to cope with the challenges of such a volatile situation, such systems need to be engineered in a manner that supports adapting to changes of the user and the environment. Obviously, such adaptivity can be realised by implementing lots of case distinctions that address individual situations. However, as the number of problems that need to be addressed increases, this will lead to bloated and unmaintainable code.

Instead, we propose the use of *components* and *reconfiguration* to achieve adaptivity. Components are considered to be black boxes, making explicit only their communication requirements by means of *required* and *provided* ports. A system is comprised from a number of components and their *configuration*, which describes how the required ports are connected to suitable provided ports. Numerous component models describe how exactly this is achieved, and how the components can communicate in order to achieve a common goal [4]. For the REFLECT project, we have developed our own component framework [6], which is specially tailored to building adaptive systems with multimedia sensors and actuators.

In a component application, individual components provide parts of the functionality required by the entire system. By substituting, adding and removing individual components, the system's behaviour can be changed. This process is called *reconfiguration*. Since entire components are replaced, little code needs to be added to the components to achieve this kind of adaptivity. Instead, it is attained on a level more coarse: the level of the system architecture.

Still, reconfiguration is a difficult problem, and while many framework support it [5] the problems often outweigh the utility. One of this problems is the necessity to plan how reconfiguration should be conducted, which usually requires both anticipation of possible future problems as well as an algorithm to figure out a reconfiguration plan that operates on a component configuration that has possibly undergone many reconfigurations already. The anticipation of future reconfiguration scenarios is a difficult task. Here, we propose a way to support the planning of reconfiguration by annotating components with assume-guarantee pairs. Informally, the guarantee describes how a component will conduct communication over its ports, given that the assumption about the communication received by its ports holds. In this paper, we introduce a semantics-based assume-guarantee framework similar to the framework proposed in [2].

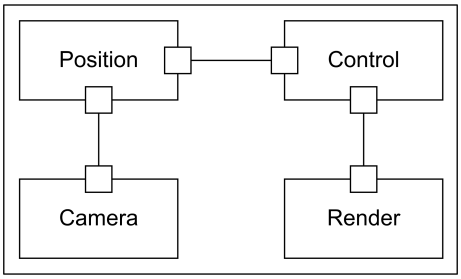

Fig. 1. Initial System

From such assume-guarantee specifications, a component's anticipation of possible employment scenarios can be derived. When designing a system, by connecting a component to other components, the composition can be checked against the guarantees provided by the communication partners, and possible mismatches can be detected. Checking composability in such a way has a long tradition (e.g., [1]). Furthermore, the intended behaviour of the entire application can be specified by assume-guarantee pairs, where the assumptions address the physical environment, and the guarantees the output produced. Again, the compatibility of a component configuration to such global specifications can be checked, and invalid application designs can be detected.

However, an invalid application configuration can still be useful under certain conditions. Identifying these assumptions and monitoring their validity allow to restrict the execution of the generally invalid application to valid situations at runtime. If the assumptions are about to get invalidated in a system run, it is still possible to execute a reconfiguration to a system that will show the desired behaviour in the new environment. The role of *monitors* is hence to allow to deploy system configurations needing assumptions that are not satisfied in the general case, and trigger reconfigurations as these assumptions get falsified.

3 Example Scenario: Adaptive Advertising

In order to illustrate our approach, we use a simple adaptive advertising scenario. The general idea of adaptive advertising is to adapt the displayed ad to the current situation in front of it – whether there are several people just passing by, a small group of persons watching the ad carefully, or just one person in front of it waiting for someone else. The system uses cameras to observe the passers-by, and by this enables the ad to react to e.g. the number of passers-by watching the advertisement, to discover their interest in the advertisement by analysing their gaze direction and exposure time, or to enable gesture-based interactions with a passer-by becoming interested in the ad.

A simple scenario within the vast ranges of possibilities the adaptive advertising setting offers is an adaptive car advertisement, in which the displayed car reacts to the position of users in front of the display: By moving around the display, a selected user controls the orientation of the car.

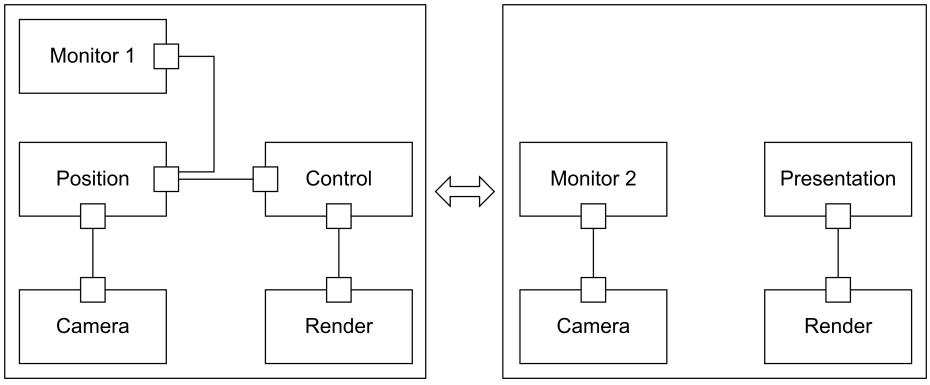

Fig. 2. System with Reconfiguration

The contract to be satisfied by this system consists of two guarantees: (G1) Being an interactive ad, the system should react to a user in front of the display. (G2) The content displayed must change at least every ten seconds: an advertising campaign using a large-scale display should not waste its capabilities by showing static content.

A first realisation of the system consists of four components (cf. Fig. 1): a camera component for image acquisition, a position detection component detecting the position of persons in front of the display, a control component selecting the person to be given control over the car movements and how his position should be related to the car's rotation, and finally a rendering component displaying content. This simple realisation is problematic, however. It cannot provide the guarantee that the displayed content changes every ten seconds, as the car movement depends on the control of a person in front of the display. Using a simple assume-guarantee calculus, we can show that the system is incomplete although it provides part of the desired functionality.

By introducing a monitor observing whether someone is in front of the display, the system can be made aware that it is about the violate its contract. Then, a reconfiguration can be triggered which alters the system such that it shows a constantly revolving car (cf. Fig. 2). In more formal terms, introducing a monitor allows to assume that the environment exhibits certain features (e.g. always have someone in front of the display) that it does not exhibit in the general case. Note that the second system (Fig. 2, right) also needs monitoring, as it again does not satisfy G1: The second system does provide interactive content to its viewers, and therefore must be changed as soon as a person is in front of the display.

In the following section, we introduce a formal framework allowing to check the properties described informally above, and show how it can be proven that the overall system satisfies the global contracts G1 and G2.

4 Assume-Guarantee Specifications

We annotate every component in our adaptive system by a pair of assertions (A, G). The assertion A formulates a property the component assumes from its environment

whereas G is the guarantee it provides to the environment given that A is satisfied. In this way every component can be proven to be correct with respect to their assume-guarantee specification.

We introduce a simple assume-guarantee framework which is formulated on the semantic domain in terms of runs over a given signature. A *signature* Σ consists of a set of provided and required ports, denoted by $ports_{prv}(\Sigma)$ and $ports_{req}(\Sigma)$ respectively. We assume the notion of a subsignature $\Sigma \subseteq \Sigma'$ and the supremum of two signatures $\Sigma \sup \Sigma'$; both notions are defined in the obvious way. A Σ-run is an abstract structure representing one possible behaviour of the system. One possibility – among others – to refine the notion of runs is to see them as capturing reception and sending of messages on ports (given by Σ) over (discrete or continuous) time.

Assumptions, guarantees as well as implementations of components are considered to be assertions. Given a signature Σ, a *Σ-assertion* E (also denoted by $E : \Sigma$) is identified with a set of Σ-runs. We assume that every Σ-assertion E can be lifted to a Σ'-assertion $E \uparrow^{\Sigma'}$ for $\Sigma \subseteq \Sigma'$. Moreover, we define the composition of assertions by $E_1 : \Sigma_1 + E_2 : \Sigma_2 := E_1 \uparrow^{\Sigma} \cap E_2 \uparrow^{\Sigma}$ for $\Sigma = \Sigma_1 \sup \Sigma_2$. From now on, signatures and liftings are omitted where they are not essential.

Assume-guarantee pairs are formulated as pairs of assertions $(A : \Sigma_A, G : \Sigma_G)$. Satisfaction is defined simply by inclusion of runs – more precisely, every run in M which is in A (i.e. satisfies A) must be in G.

Definition 1. *Let $M : \Sigma$ be an implementation, and $A : \Sigma_A$, $G : \Sigma_G$ two assertions. M satisfies $(A : \Sigma_A, G : \Sigma_G)$, denoted by $M \models (A, G)$, if and only if $\Sigma_A, \Sigma_G \subseteq \Sigma$ and $M \cap A \subseteq G$.*

Parallel composition of implementations preserves this satisfaction relation.

Lemma 1. *Let $M_1 : \Sigma_1$, $M_2 : \Sigma_2$ be two implementations, and let $ports_{prv}(\Sigma_1) \cap ports_{prv}(\Sigma_2) = \emptyset$ and $ports_{req}(\Sigma_1) \cap ports_{req}(\Sigma_2) = \emptyset$. If $M_1 \models (A_1, G_1)$ and $M_2 \models (A_2, G_2)$ and A is an assertion for which it holds $A \cap G_1 \subseteq A_2$, $A \cap G_2 \subseteq A_1$, and $A \subseteq A_1 \cup A_2$ holds, then $M_1 + M_2 \models (A, G_1 \cap G_2)$.*

When building component systems we want to check whether the resulting specification satisfies a global system specification. Therefore we introduce a refinement relation which allows assumptions to be weakened and guarantees to be strengthened.

Definition 2. *(A, G) refines (A', G'), denoted by $(A, G) \preceq (A', G')$, if $A' \subseteq A$ and $G \subseteq G'$.*

A major requirement for refinement relations is its compatibility with the satisfaction relation for implementations, i.e. whenever an implementation satisfies a refined contract, it satisfies the original contract.

Lemma 2. *If $M \models (A, G)$ and $(A, G) \preceq (A', G')$ then $M \models (A', G')$.*

A global system specification $(A_{\mathrm{sys}}, G_{\mathrm{sys}})$ can then be verified in the following way. Assume that the composed system $M_1 + \ldots + M_n$ satisfies (A, G), then in order to show that it also satisfies the global system specification $(A_{\mathrm{sys}}, G_{\mathrm{sys}})$ it suffices to show $(A, G) \preceq (A_{\mathrm{sys}}, G_{\mathrm{sys}})$.

In order to verify dynamic, reconfigurable systems, we must define the notions of a *composable* system that is *configured* at each moment by *connectors*. From now on, we consider more concrete runs of the form $\rho : \mathbb{R}_0^+ \to S$ with S a domain for states of runs.

Definition 3. *A set of implementations $M_1, \ldots, M_n$ is called* composable *iff $\Sigma_i \cap \Sigma_j$ is the empty signature for all $i \neq j$. A* configuration *of $M_1, \ldots, M_n$ is a set of runs C over $\Sigma \cup Con(\Sigma)$, $\rho : \mathbb{R}_0^+ \to \mathcal{P}(\Sigma \cup Con(\Sigma))$, where $\Sigma = \sup_{i=1}^{n} \Sigma_i$ is the supremum over all signatures, and $Con(\Sigma) = ports_{req}(\Sigma) \times ports_{prov}(\Sigma)$ is the set of all connectors. A configuration is* valid *iff for all $\rho \in C$. $(r, p) \in \rho(i)$ implies that $r, p \in \rho(i)$ or $r, p \notin \rho(i)$.* [1] *The composition under C, $(M_1 + \ldots + M_n)|_C$ is the set $C \cap (M_1 + \ldots + M_n) \uparrow^{\Sigma \cup Con(\Sigma)}$.*

Lifting $M_1 + \ldots + M_n$ to $\Sigma \cup Con(\Sigma)$ means the set of all runs in which each state of a run $\rho \in M_1 + \ldots + M_n$ was extended with each subset of connectors $Con(\Sigma)$.

A composable system is therefore a set of implementations that do not share any ports a priori, hence allowing a configuration to define the port connections through its runs (note that we consider only *valid* configurations in the following). Then, a composition under configuration is the set of runs that are compatible with the connections defined by a run in the configuration.

Example 1. We give a very small example for a composable system and a configuration in the following. Let $ports_{req}(\Sigma_1) = \{a\}, ports_{prv}(\Sigma_1) = \emptyset$ and $ports_{prv}(\Sigma_2) = \{b\}, ports_{req}(\Sigma_2) = \emptyset$ be two signatures. Let $M_1 = \{\rho_1, \rho_1'\} : \Sigma_1$ and $M_2 = \{\rho_2\} : \Sigma_2$ be properties such that

$$\rho_1(i) = \begin{cases} \{a\} \text{ if } 0 \leq i \leq 2 \\ \emptyset \quad \text{otherwise.} \end{cases} \qquad \rho_2(i) = \begin{cases} \{b\} \text{ if } 1 \leq i \leq 3 \\ \emptyset \quad \text{otherwise.} \end{cases}$$
$$\rho_1'(i) = \emptyset \text{ for all } i.$$

As the signatures of M_1 and M_2 are disjoint, M_1 and M_2 are composable. Let hence C be a configuration of M_1 and M_2 containing all ρ_c for which it holds that $(a, b) \in \rho_c(i)$ for $2 \leq i \leq 3$. Then, the composition $M = \{\rho\} = (M_1 + M_2)|_C$ consists of the single run ρ such that

$$\rho(i) = \begin{cases} \{a\} & \text{if } 0 \leq i < 1 \\ \{a, b, (a, b)\} & \text{if } 1 \leq i \leq 2 \\ \{b\} & \text{if } 2 < i \leq 3 \\ \emptyset & \text{otherwise.} \end{cases}$$

ρ is the composition of the runs ρ_1 with ρ_2 under one $\rho_c \in C$. Note that ρ_1' is not part of the composed system, as there is no run in M_2 to which it is compatible.

Note that Lemma 1 is still valid for composition under configuration, as the set of implementations is further constrained. It is also possible to take the configuration runs into the guarantee, as can be seen easily.

[1] In a more general setting we would require that the r and p are equal in value. Here however, we only consider predicates.

We now apply the described approach to our example scenario of Sect. 3. First, we must refine the general assume-guarantee framework to a real-time LTL logic given as follows.

Definition 4. *The set of RT-LTL-formulae is inductively defined by the grammar*

$$A ::= true \mid p \mid p_1 \sim p_2 \mid \neg A \mid A \vee A \mid \Box_t A$$

with ports $p, p_1, p_2 \in \Sigma$. Let $\rho : \mathbb{R}_0^+ \to \mathcal{P}(\Sigma \cup Con(\Sigma))$ be a valid run over Σ and $Con(\Sigma)$. Then $\rho \models A$ iff for all $i \in \mathbb{R}_0^+$. $\rho, i \models A$. The satisfaction relation $\models$ between pairs ρ, i and RT-LTL-formulae A is defined as follows.

1. $\rho, i \models true$.
2. $\rho, i \models p$ *iff* $p \in \rho(i)$.
3. $\rho, i \models p_1 \sim p_2$ *iff* $(p_1, p_2) \in \rho(i)$.
4. $\rho, i \models \neg A$ *iff* $\rho, i \not\models A$.
5. $\rho, i \models A \vee B$ *iff* $\rho, i \models A$ *or* $\rho, i \models B$.
6. $\rho, i \models \Box_t A$ *iff* $\forall i \leq j \leq i + t.\ \rho, j \models A$.

In the following, we use $\Box A$ as abbreviation for $\Box_\infty A$. $\Diamond_t A$ and $A \Rightarrow B$ are defined as usual. Note that $\Box A$ is equivalent to A.

Example 2. The system contract that must be shown for the system is the tuple

$$(GI, (\Box\Diamond_{10} CI) \wedge (\Box_{0.5} PT \Rightarrow \Diamond_1 R)).$$

Here, GI is "the camera produces a good image", CI "the image shown on the display changed", PT is "there is a person in front of the ad", and R stays for "the system responded to the person in front of the ad". Following Lemma 2, we must show that the contract guaranteed by the composition, (A, G), is a refinement of the system contract (A_{sys}, G_{sys}). For the sake of brevity, we will focus on discussing the system guarantee $\Box\Diamond_{10} CI$. In the example, the guarantee G of the composition (cf. Fig 2) can be shown to guarantee the following five properties, where C_1 is a shorthand notation for a conjunction of connection constraints defining "the system is in configuration 1" (cf. Fig. 2, left), and C_2 denotes similarly "the system is in configuration 2" (cf. Fig. 2, right).

1. $C_1 \Rightarrow PT \Rightarrow CI$
2. $C_2 \Rightarrow CI$
3. $C_1 \Rightarrow (\Box_3 \neg PT) \Rightarrow (\Diamond_{3.7} C_2)$
4. $C_2 \Rightarrow (\Box_{0.5} PT) \Rightarrow (\Diamond_{0.7} C_1)$
5. $C_1 \vee C_2$

Note that property 3 and 4 express changes in configurations: property 3 expresses that the first configuration (reacting to viewers) will be reconfigured in 3.7 seconds to the second configuration (showing animated content) if the scene in front of the display stays empty for three seconds. This property can be derived from the guarantee of Monitor 1 together with the connection properties of configuration one. Similarly,

property 4 expresses that the system will be changed within 0.7 seconds if a person is seen in front of the display for at least 0.5 seconds. Note that obviously, property 3 to 5 are derived from the behaviour of monitors and the specification of the configuration.

Properties 1 and 2 on the other side describe the behaviour of the system under each configuration: in configuration 1 (reacting to viewers), the displayed image changes if a person is in front of it. In configuration 2 (showing animated content), the image always changes.

With the help of a small realtime-LTL calculus, then, we can to show that these five properties imply $\Box\Diamond_{10}CI$, which is one step in showing the refinement $G \subseteq G_{sys}$. Altogether, it can be shown that the composition under the given reconfiguration refines the global contract. Hence, by switching between two initially insufficient configurations, the system is indeed showing the desired global behaviour.

5 Conclusion

Building pervasive user-centric applications brings several challenges, as they are operating in highly dynamic and uncertain environments. In this paper, we took a closer look in how this challenge can be taken by using a simple assume-guarantee-framework. The assume-guarantee approach allows to make assumptions of a configuration about the environment explicit. By monitoring these assumptions, it is possible to deploy an application that does not satisfy its contract in the general case, but only under given assumptions. As soon as these assumptions are violated, the system is reconfigured, so that the new configuration satisfies the contract under the now given assumptions.

An interesting future work is the introduction of probabilistic assume-guarantee contracts, as presented in [2]. Pervasive user-centric applications interface with the real world through sensors and actuators, which may be unreliable in and exhibit not only non-deterministic, but also probabilistic behaviour. With a probabilistic assume-guarantee framework, it would be possible to model this uncertain behaviour of the environment, and reason about the performance of pervasive user-centric applications in these environments.

References

1. Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *8th European software engineering conference (ESEC '01)*, pages 109–120. ACM Press, 2001.
2. B. Delahaye and B. Caillaud. A Model for Probabilistic Reasoning on Assume/Guarantee Contracts. *ArXiv e-prints*, November 2008.
3. Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.*, 5(4):596–619, 1983.
4. K. Lau and Z. Wang. Software component models. *IEEE Transactions on Software Engineering*, 33(10):709–724, 2007.
5. M. Sadjadi and P. McKinley. A survey of adaptive middleware. Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, 2003.
6. Andreas Schroeder, Marjolein van der Zwaag, and Moritz Hammer. A Middleware Architecture for Human-Centred Pervasive Adaptive Applications. In *2nd Int. Conf. on Self-Adaptive and Self-Organizing Systems (PerAda '08)*, volume 0, pages 138–143, Los Alamitos, CA, USA, 2008. IEEE Computer Society.