On Runtime Service Quality Models in Adaptive Ad-hoc Systems

Christian Peper
Fraunhofer IESE
Fraunhofer-Platz 1
D-67663 Kaiserslautern
[+49] 631 6800-2219
christian.peper@iese.fraunhofer.de

Daniel Schneider
Fraunhofer IESE
Fraunhofer-Platz 1
D-67663 Kaiserslautern
[+49] 631 6800-2187
daniel.schneider@iese.fraunhofer.de

ABSTRACT
Ad-hoc computer systems can automatically realize higher services when at least two distributed and communicating (embedded) devices come together. For this purpose, they must be able to manage appearance and loss of devices and resources, and they have to adapt to changes in requirements and environment. Based on a component-oriented approach for adaptive ad-hoc systems, this paper suggests a high-level service quality reference model to advocate further research on the quality matching problem between service provider and client components.

Categories and Subject Descriptors
C.2.4 [Distributed Systems], C.3 [Special-purpose and Application-based Systems], D.2.2 [Design Tools and Techniques], D.2.11 [Software Architecture], D.2.12 [Interoperability]

General Terms
Design, Reliability

Keywords
ad-hoc systems; adaptivity; ambient intelligence; distributed systems; ubiquitous computing; component-orientation; quality-of-service

1. INTRODUCTION
The number of distributed, embedded computing devices around us increases year by year. Ambient intelligence systems, for example, are especially characterized by their ubiquitous and non-invasive presence throughout the physical environment of human users [1]. Such systems usually contain devices and services that might drop in and out during runtime. If a system is able to launch higher services and applications whenever appropriate resources meet, it is called an ad-hoc system (following the term ad-hoc network [3][2]), provided the necessary infrastructure is fully distributed and does not rely on centralized constituents.

Furthermore, such systems have to react on dynamic changes imposed by user demands or the state of their environment. All changes in functional and quality-of-service (QoS) requirements and in the availability of resources should be suitably handled. Scarce resources like memory, processing power, or communication bandwidth must be managed at runtime in order to cope with volatile quality requirements and concurrent use by different clients. Since higher services are built upon basic services, it is common that these higher services are degraded in their service quality or even disappear, whenever one of the used basic services fails and no substitute is available. Systems that are able to autonomously cope with these issues are called adaptive.

Adaptive ad-hoc systems aim at providing an optimized service depending on the availability of services, resources, and environmental state. To achieve this goal, certain design decisions that are usually taken at development time, must be delayed and are resolved at runtime. This implies the existence of corresponding runtime models and mechanisms. In hierarchically organized component systems (like KobrA [15]), these ideas should not only be applied to the system as a whole. If they are considered for each participating component, more flexible designs can be realized. This means that a component has to manage and communicate knowledge about:

- the current physical environment into which the component is embedded (e.g., situation of the users),
- the service variants that it is able to provide, and
- the possible realization variants including knowledge about the availability of required resources and how to determine the best configuration for a certain situation.

This knowledge is used by two adaptivity support tasks: matching and mapping. Matching covers the negotiation of service functionality and quality between a service providing component and its client. This is based on the exchange of a type variant specification at runtime. To satisfy such a specification, the mapping task uses runtime realization models for acquiring and composing suitable resources.

This paper focuses on the matching task. It suggests an abstract model how to generally characterize service and quality variants as well as the opposing interests of service provider and client. However, it does not suggest concrete quality models or matching algorithms, which strongly depend on the application domain. Furthermore, the paper shortly discusses problems in the context of resource-constrained embedded systems that do not have the
2. COMPONENT-BASED AD-HOC SYSTEMS

Component engineering principles have been shown to foster the development of well-structured and also (re-)configurable static systems. Therefore, it is not surprising that component techniques were also successfully used to move certain system construction tasks (like service discovery and binding) from development time to runtime. Furthermore, component systems can also support a dynamic realization of the matching task. To illustrate how, we will shortly revisit the approach introduced in [2] and summarize the main building blocks.

The ad-hoc systems considered here simply consist of more or less movable physical devices, which are capable of communicating with some of their neighborhood devices. Devices contain specific resources offered to the context through dedicated software components, the providers. A provider is exclusively responsible for offering its resources as services of a specific type. It creates new service instances (so-called service units) on demand when requested by a client. The provider owns all realization knowledge for its service units, and in turn asks other providers for the needed resources (service units). A service unit can be regarded as a logical, potentially distributed component that realizes a certain functional relation (the service) between its functional interface ports. Ports are the unit’s atomic, i.e. non-distributed, service access points. With the exception of “read-only” services, a service unit only serves exactly one client to avoid access conflicts and side-effects.

![Figure 1. A service unit with control port](image)

As argued above, certain development tasks in ad-hoc systems are moved to runtime execution. Such tasks include the selection of available resources, the negotiation about quality-of-service, the binding to concrete object instances, etc. For this purpose, each service unit is extended by a control port that allows some manipulation of the unit-specific functional and non-functional properties. Via this control port, client and service unit can negotiate and fine-tune all settings specific to their individual client-server relation. No other clients have access to the control port of the same service unit. Figure 1 shows a service unit with functional ports and the control port, which is sub-structured into six channels.

The mode channel provides control over the operational modes such as starting, stopping, resetting, etc. The state channel allows saving and restoring the service unit’s current abstract state. The watchdog channel allows mutual surveillance of the presence of both client and service unit, because these components might disappear without any prior notification. The index channel deals with explicit internal and environmental runtime knowledge. The configuration channel supports the adaptation of a component service to changed requirements and available resources. It considers the functional appearance comprising offered features and port structure. The configuration channel is divided into required and provided functionality. When an important resource is lost, the service unit can decide to degrade the provided functionality, e.g., by withdrawing access to a feature. On the other side, the client could decide to reduce its claims and require a less restrictive functional type in the same way. Finally, the quality channel is also twofold: it is used to exchange information about the required quality desired by a client and the provided quality currently offered by the service unit. In the following, we will focus on the meaning of configuration and quality channels.

3. SERVICE, FUNCTIONALITY, AND QUALITY

To advertise their offered services, providers usually publish their service type, and clients look up service units based on the desired type. This will work for ad-hoc systems, if the developers of provider and client have the same understanding of the type, i.e., there exists a static type specification at development time. In an adaptive system, however, a component’s service type is subject to changes at runtime. Hence, the static type specification should describe a whole family of possible service variants rather than only one. The configuration and quality channels help components to exchange the corresponding runtime descriptions of available or required variants. But these descriptions do not necessarily contain a complete and explicit specification of the considered service type variant. Instead, a selector of the variant can be sufficient to exactly indicate the variant’s properties, provided the static specification of the used type family was available to the developers of both provider and client component. At runtime, components will not have to deal with any explicit knowledge about a service type.

A service type variant is statically characterized by its interface and behavior with functional and non-functional aspects as depicted in Figure 2. In the presented approach the interface consists of a number of (behavior-free) ports of different port types. The service behavior only becomes evident at these ports. Switching from one variant to another could result in the addition or removal of ports. If, for example, the video transport feature in Fig. 2 is activated, the corresponding ports videoIn and videoOut will be made available. Because component ports are usually connected to realize higher services, any changes in the port structure necessitate a (partial) re-organization of the component architecture.

This background suggests dividing the adaptivity management into two levels: the adaptation of functionality including port structure and functional behavior, and the adaptation of service quality considering the non-functional aspects of a given functionality. This distinction has two advantages: (1) As long as a concrete functionality is configured, no changes in ports and functional behavior will occur. Consequently, there is no need to modify the design of the realization context into which the considered component is embedded by port connections. However, an adaptation of the service quality is still possible. (2) The exact nature of the service quality normally depends on the configured functionality. Therefore, it is reasonable to firstly...
select the functionality and to consider the related quality afterwards. In the example we have to decide on using the video transport function before we can talk about corresponding quality attributes like delay or resolution.

Figure 2. Overview of terms

**Functionality.** A component’s functionality or functional configuration \( F \) is characterized by a constant set of functional ports, through which the component offers its service, and by a fixed behavior at these ports. Whenever functional behavior or port set change, the component is said to be reconfigured to another functionality. For a given service type \( T \) (representing a family of variants), the functionality space \( \text{FS}_T \) is the set of all functionalities admitted by \( T \)’s type specification. A functionality range is a subset of the functionality space.

**Quality.** In a similar way, the service’s quality can be characterized. However, the nature of the quality space does not only depend on the service type, it depends on the considered functional configuration \( F \). A component’s quality configuration (or short quality) \( Q \) reflects the quality-of-service offered by the underlying functionality. Each functionality \( F \) therefore induces its own, specific quality space \( \text{QS}_F \) of all possible quality configurations. A quality range is a subset of the functionality’s quality space.

**Service Configuration.** To yield an overall configuration of the service \( T \), it is now possible to combine a functional configuration and a belonging quality configuration into a service configuration or service variant \( C = ( F, Q, \mathcal{F} ) \) with \( F \in \text{FS}_T \) and \( Q \in \text{QS}_F \). Accordingly, the service configuration space \( \text{CS}_T = \{ ( F, Q ) | F \in \text{FS}_T \land Q \in \text{QS}_F \} \) is the set of all such service configurations.

### 4. STRUCTURED SERVICE CONFIGURATIONS

Experience shows that it is usually possible and straightforward to sub-structure a service into features (like VideoTransport (VT), AudioTransport (AT), NormVolume (NV) in Fig. 1). This approach supports the separate specification of each feature. Once again, each feature includes functional and non-functional aspects (cf. right part of Fig. 2).

The feature’s function (e.g., VideoTransport) is acting on a constant and finite set of ports \( \text{videoIn} + \text{videoOut} \) and has a constant (non-adaptive) behavior with certain non-functional properties. Furthermore, the list of available functions \( \mathcal{F}_T \) for a given service type \( T \) is expected to be a finite and constant set. Based on functions, a functionality \( F \) can then be defined over the inclusion of currently available functions, i.e., \( F \subseteq \mathcal{F}_T \). If, for example, the multimedia service offers the three functions \( \mathcal{F}_{\text{MultimediaChannel}} = \{ \text{VT}, \text{AT}, \text{NV} \} \), then the subset \( F = \{ \text{AT}, \text{NV} \} \) is a functionality of service type MultimediaChannel.

A function’s non-functional properties are characterized by quality attributes (like sample rate and image size in the video example). Each quality attribute \( q \) has its specific domain \( \mathcal{D}_q \). We demand that the quality attribute list \( \mathcal{Q}_f \) is fixed for a given function \( f \). Thus the quality space for this function is the set product of its attribute domains \( \text{QS}_f = \prod_q \mathcal{D}_q \). Typical quality spaces are spanned by a variety of QoS attributes [12]. We can distinguish between standard attributes that have to be taken into account by each functionality, and specific attributes that only make sense for the respective functionality. Functionality-specific attributes typically consider non-functional parameters describing performance issues like the sampling rate of a sensor or the computation delay between input and output. Examples for standard quality attributes are port location(s), durability, prices, reliability, security, etc.

Accordingly, a feature combines a function with several quality attributes. Because a service variant consists of a functionality (now given as a set of functions) and a quality (now given as a set of attributes for these functions), it is also possible to regard the service variant as a set of features (each consisting of a function with corresponding attributes). The following table shows exemplary functions and their relation to different, but also to common attributes. The attribute audio volume, for example, is relevant for both AT and NV function.

<table>
<thead>
<tr>
<th>Functions FL</th>
<th>Quality attributes QL</th>
<th>location in port</th>
<th>location out port</th>
<th>video sample rate</th>
<th>image size</th>
<th>audio sample rate</th>
<th>audio volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Transport (VT)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Audio Transport (AT)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Norm Volume (NV)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Hence the quality of a service variant does not need to be specified explicitly, it can be implicitly derived from the attribute domains of those features contained in the corresponding functionality. Note that features are generally not independent: they may use the same ports and have priorities or other functional and non-functional restrictions concerning their concurrent use (e.g., volume normalization NV requires a running audio stream AT). For this reason they are not offered as independent service units with different types.

### 5. THE MATCHING PROBLEM

As argued, client and provider of the service have to agree about a service configuration that is acceptable for the client and realizable by the provider. For this service level agreement, the client must be able to specify which configurations he requires, and the provider must specify which configurations are provicable. To clarify this situation, we assume that both provider and client share the same perception of a configuration space from which the service configuration is taken. The client expresses certain requirements that define a subset \( \text{CR} \) of the service configuration space, i.e., the configuration range \( \text{CR} \subseteq \text{CS}_T \) of required configurations. Figure 3 illustrates a configuration space with this set of acceptable configurations. On the other hand, the provider will generally not be able to provide...
all configurations from the whole configuration space. Only a certain subset $\mathcal{CP}$, the set of providable configurations, can actually be offered to the user. Nevertheless, as long as the provided configuration $c_p$ (represented by the cross) is within the range $\mathcal{CR}$ of required configurations, the user will be satisfied with the service. Therefore, we say that the required and provided configuration ranges match, if their intersection is non-empty, i.e., $\mathcal{M} = \mathcal{CP} \cap \mathcal{CR} \neq \emptyset$. The intersection $\mathcal{M}$ itself is called a matching.

![Figure 3. Configuration space](image)

Actually, it is sufficient if we choose any location from within the intersection of $\mathcal{CP}$ with $\mathcal{CR}$, since this configuration can be provided and satisfies the user requirement. However, a client will usually have preferences concerning the exact nature of a configuration within the required configuration range, because some of the features or quality attribute values might be more useful to him than others. In particular there might be one optimal required configuration $c_{ropt}$ (indicated by the square in the middle of $\mathcal{QR}$), from which the client will benefit most.

On the other hand, the provider also has certain configurations he would prefer to offer and possibly an optimal provided quality $c_{popt}$. This is usually caused by the different amount of resources the provider has to invest to realize the respective configuration. It would be perfect, if both optimal configurations, i.e., required and provided, were the same. But due to opposed interests (best service vs. few resources), this will almost never be the case.

Therefore, an approach is needed to identify not only any matching configuration, but also a good one that is as useful as possible for both user and provider. As a solution to this problem both parties could state for each service configuration in $\mathcal{CS}$, how useful this configuration is. Such an evaluation of the configuration’s utility can be modeled by a function from the configuration space to some evaluation domain: A function $u : \mathcal{CS} \rightarrow U$ that maps service configurations to utility values, is called utility function, if $U$ is a totally ordered domain of utility values, where higher values represent a better utility (below we will assume $U = [0,1] \subseteq \mathbb{R}$ ). In addition, some “neatness” condition should ensure a suitable distribution of utility values to avoid the following way of misuse: all configurations with the exception of the optimal one are mapped to the lowest utility value in order to get higher chances to receive this best one.

Now both provider and client interests can be defined with the two utility functions $u_P : \mathcal{CP} \rightarrow U$ returning the provider view, and $u_R : \mathcal{CR} \rightarrow U$, returning the client (requirement) view. Figure 4 sketches such a pair of utility functions on basis of a one-dimensional configuration space. As already in Figure 3, the configuration $c_{popt}$ with the best provider evaluation (circle) is different from the configuration $c_{ropt}$ with the best client evaluation (square).

![Figure 4. Utility and fairness](image)

This raises the question, what a “good” matching could be. Suppose, for example, two configurations had utility ratings of $u_P(c_1)=0.5$ and $u_R(c_1)=0.9$, as well as $u_P(c_2)=0.6$ and $u_R(c_2)=0.6$. The fairest solution is obviously to choose $c_2$ with its 0.6/0.6 utility values, but the best global benefit would result from the 0.5/0.9 ratings of $c_1$. But in the latter case the provider is tranquilized with a utility of 0.5, although 0.6 would be achievable.

Apparently, it seems to be good when (1) the utilities’ mean value (or equivalently their sum) is as high as possible, because this optimizes the overall utility, and (2) the distance of both utility values is as low as possible, since this appears to be a fair compromise. In Figure 4, the most useful configuration $c_{mean}$ offers the maximum of the mean values of both $u$-functions (represented by the smaller grey graph in the middle). However, it comes with a significant disadvantage for the client. On the other hand, the best fair matching is characterized by the intersection points of both utility function graphs. But it only contains the single configuration $c_{opt}$. Matchings with only one configuration are problematic, because they offer no scope for the provider to switch to another configuration in case of resource changes, i.e., they do not support adaptive behavior without re-negotiation.

Therefore, a good matching should (3) also contain as many configurations as possible. Following these three observations, we can define two matching approaches with focus on (1) and (2), respectively.

The best mean utility value $u_{mean}$ is the maximum of all utility mean values for which there exists a common configuration. The best $d$-mean matching $M_{d,mean}$ emphasizes (1) and contains all configurations with utility values lying within a maximum deviation of $d$ around the best available mean value:

$$u_{mean} = \max_{c \in \mathcal{M}} \frac{1}{2} \cdot (u_P(c) + u_R(c))$$

$$M_{d,mean} = \{ c \in \mathcal{M} | \frac{1}{2} \cdot (u_P(c) + u_R(c)) \leq d \land |u_R(c) - u_{mean}| \leq d \}$$
Secondly, the best d-fair matching focuses on a “relaxed” fair compromise in the sense of (2), defined in a similar way (see also Fig. 3):

\[ u_{\text{fair}} = \max_{c \in \mathcal{M}} \{ u \in \mathcal{U} \mid u_P(c) = u_R(c) = u \} \]

\[ \mathcal{M}_{\text{d-fair}} = \{ c \in \mathcal{M} \mid |u_P(c)-u_{\text{fair}}| \leq d \land |u_R(c)-u_{\text{fair}}| \leq d \} \]

Furthermore, there could be situations, where the client abandons his interests in favor of an optimal provided configuration \( c_{\text{opt}} \), and vice versa. Currently, we expect that there is no universal solution how to derive a matching from provided and required utility values. A pragmatic solution could be to let client and provider also negotiate the corresponding comparison policy. The policy parameter then should at least contain support for the discussed matching variants best provided, best required, best fair, and best mean, as well as for the deviation values.

So far, we did only consider utility functions that are evaluating complex configurations. But if a configuration is structured into functionalities and quality aspects, as suggested in the previous section, it becomes possible to evaluate such a singular aspect (or a subset of aspects) by a dedicated pair of utility functions. Such a set of partial u-functions can be easily integrated into one global function by computation of a weighted sum or counting the aspect winners, etc. Thus decomposing into features can facilitate the construction of a global utility function.

6. EXAMPLE: VIDEO STREAMING

The following example focuses on quality attributes and considers a communication provider that offers specialized video connection channels as service units. The functionality space is the simplest possible, because only one function is contained. However, the consideration of some video-specific quality attributes leads to an interesting relation between required and provable quality ranges. The service quality space is characterized by two parameters: the frame rate \( f_r \) measured in frames per second in the range from 0 to 100 frames/s, and the image resolution \( r_e \) measured in mega pixels in the range from 0 to a maximum of 2.2 million pixels (HDTV at 1920x1080 has 2.07 Mpx).

The video connection quality is required to be not smaller than 0.44 Mpx (old PAL, 768x572) and not slower than 10 frames per second. The optimal quality from the client’s viewpoint is obviously the maximum of both image size and frame rate, providing the best video connection. Once again it is indicated as a square in Figure 5. Aiming at a utility domain ranging from 0 to 1 within \( QP \), the required utility function for the considered quality aspects could be defined as

\[ u_R(f_r, r_e) = 0.5 \cdot (((f_r - 10)/90) + (r_e - 0.44)/1.76) \cdot \]

On the other hand, this video service might be realized based on a communication system with limited bandwidth, and on a standard video compressor reducing the pixel storage from 24 bit / pixel to 1.0 bit / pixel (higher compression rates are typically 0.4-0.8 bit/px). The communication system has an upper bandwidth limit for payload of about 65 Mbit/s, meaning that it is able to transport an HDTV 1080p30 video stream with 2.07 Mpx/frame \( \cdot \) 1.0 bit/px \( \cdot \) 30 frames/s = 62.2 Mbit/s. This upper limit in the provided qualities is characterized by a hyperbola representing the constant 65 Mbit/s.

At the other end, we assume the communication system to be designed to send at least 1 kiB packets every 1 ms: due to some internal technical reasons this is the smallest resource that can be assigned to a communication channel. Therefore the minimum provable video quality is actually (0 Mpx, 0 frames/s), but even in this case, the used bandwidth is 1024 \( \cdot \) 8 bit \( \cdot \) 1000 s\(^{-1}\) = 8.2 Mbit/s. From the provider viewpoint, the most useful video quality must lie on the 8.2 Mbit/s curve, because higher qualities consume more resources, but lower qualities do not use less. Since it is more efficient to seldom receive small images, even if the resulting data rate is the same, the optimal provable quality is located at the upper end of the 8.2 Mbit/s curve. The provider utility function could therefore be defined like

\[ u_D(f_r, r_e) = 1 - ((f_r \cdot r_e - 8.2) / (65-8.2)) \cdot \]

With these functions, a fair matching is given by the condition \( f_r \cdot r_e + 0.284 \cdot f_r + 12.91 \cdot r_e - 65 = 0 \) (dashed line in the figure). Functional analysis of the mean value function yields an extremum at (16.13 / 0.33). Unfortunately, this point is neither part of the intersection \( M = QP \cap QR \) of acceptable qualities, nor is it a maximum in both dimensions. At least it is clear that \( M \) does not contain any further local extreme value. The best utility values must therefore be located on the border line of \( M \) – actually it is \( u_{\text{mean}} = 0.629 \) so that \( M_{\text{mean}} = \{ (10, 2.2) \} \), and \( u_{\text{fair}} = 0.532 \) with \( M_{\text{d-fair}} = \{ (15.8, 2.2) \} \).

We can learn from this example that the determination of a good matching can become really difficult, because provided and required quality spaces and their utility functions might be of a completely different nature. Additionally, utility functions are time-dependent so that the functional analysis (if applicable at all) must be performed at runtime and is often not straightforward.

7. IMPLEMENTATION ISSUES

Because the negotiation of functionalities and qualities is intended to run also on smallest embedded devices, there is a need for efficient representation and implementation. In particular for tiny applications, it does not make sense when the computation of qualities consumes more resources than the application itself. The example showed that - even for simple systems - the complexity of a matching can dramatically increase. Hence, an algebraic solution appears to be impossible for the intended small systems. But also the potential computation effort in a numeric approach
The utility areas for the user are chosen a little bit arbitrarily, utility is directly related to the bandwidth needed for the service. Remaining bandwidth can be used for other users. Therefore, the 0.2, 0.4, 0.6, 0.8, and 1.0 respectively. For the provider, it is for provider and user, both partitioned into five areas of utilities intervals. Figure 6 shows the utility functions from the example utility evaluation domain \( U \) into a discrete and small number of arrays could be an appropriate and straightforward way to exchange functionality configurations. Most service providers in our context are expected to have even less than ten features.

Discretization of the Utility Domain  To further reduce the matching computation effort, it can be reasonable to divide the utility evaluation domain \( U \) into a discrete and small number of intervals. Figure 6 shows the utility functions from the example for provider and user, both partitioned into five areas of utilities 0.2, 0.4, 0.6, 0.8, and 1.0 respectively. For the provider, it is apparently most useful not to consume bandwidth, because remaining bandwidth can be used for other users. Therefore, the utility is directly related to the bandwidth needed for the service. The utility areas for the user are chosen a little bit arbitrarily, having the following idea in mind: the lowest quality has less than 25 frames/s, because it would result in a flickering video connection. Having more resources available, it is therefore reasonable to invest them primarily into a higher frame rate. But even highest frame rates are not reasonable as long as a good resolution is missing. Therefore the next step is to increase the resolution to 1.0 Mpx (1152x864) or more, then to increase again the frame rate to 50 Hz or more, and finally to increase both frame rate and resolution to more than 60Hz / 1.3 Mpx (1280x1024). Keep in mind that this video utility model can be more sophisticated, but this one illustrates two issues: (1) The utility functions of user and provider can be quite complementary. (2) Even simple models can already provide realistic utility functions. (For the description of the user function, only two coordinates in the quality space per area are needed.)

Discretization of Quality Aspect Domains  In a similar way, it is reasonable to reduce the number of different quality values, by admitting only a small number of intervals or even only some fixed values, e.g., standardized video resolutions. In the end, we can record that one core problem of quality matching is to efficiently describe and process sub-areas of the configuration/quality space and their intersections.

8. RELATED WORK
The following section is divided into one part considering related approaches towards adaptivity and models at runtime, and another part considering quality of service specifications.

Adaptivity and Runtime Models  Recently numerous projects have emerged in the areas of ubiquitous computing [1] and ambient intelligence. As dynamicity is an inherent characteristic of these domains, some of the projects had a detailed look into dynamic adaptation and models at runtime. The MADAM project [6] follows an architecture centric approach where dynamic adaptation is realized in an application independent middleware. The core idea is to transfer software product line concepts, usually used for design time variability management, into run time. Consequently, there is a run time representation of the applications architectural models (including the variability) which serves as a basis for an adaptation manager component to reason about and to control adaptation. The successor project MUSIC [7] will extend and generalize the experimental solutions developed in MADAM. Further related projects are Robocop [19], Space4U, and Trust4All [8][9]. These projects, which followed up each other, had and have the goal of constructing an open configurable middleware for consumer devices, focusing on the provision of extra-functional properties, such as power awareness, resource awareness, fault management, software management and trust. A rather new project dealing with dynamic adaptation is the DivA project [13]. Diva’s goal is to develop a new tool-supported methodology with an integrated framework for managing dynamic variability in adaptive systems based on techniques from the domains of aspect-orientation, model driven engineering and software product lines.

All of the above approaches rely on specific middleware and/or centralistic entities. As AmbiComp is focused on small embedded devices, which are attached to every day objects, adaptivity models and mechanisms should be light weight and completely distributed. Therefore we can not simply use or tailor one of these approaches for the AmbiComp project. Nevertheless, there are many interesting aspects within these projects which helped us developing our approach.

Service and Quality Specification  Prerequisite to the dynamic negotiation of QoS contracts is an explicit and computable specification of a service’s quality. As of now, we investigated what kind of information shall be part of an AmbiComp QoS specification. However, it is yet to be determined how exactly this information is to be modeled. Therefore we had a brief look at existing QoS specification languages, mostly stemming from the domain of communication systems.
We found the taxonomy for QoS specification languages by Nahrstedt and Jin to be a good starting point. It gives a nice overview and classification of the most prominent QoS specification languages [11]. A further very interesting language is the Component Quality Modelling Language (CQML), which has been inspired by QML and was developed by Aagedal in the course of his PhD work [14]. A UML-based generative approach for QoS components is presented in [18].

9. CONCLUSION
Based on a component-oriented approach for the development of adaptive ad-hoc systems, we have introduced and illustrated a high-level reference model for the matching problem in adaptive systems. It describes several aspects of the reconciliation of interests between service provider and client at runtime. The presented ideas emerged from ongoing project work. First observations indicate that – in particular for small systems with few computational resources – a significant amount of research will be necessary to elaborate efficiently working solutions. In particular, this includes the three interrelated areas:

1. determination and exchange of (discrete or other) configuration space models at runtime,
2. efficient computation of space intersections in order to determine matchings, and
3. the construction support for utility functions at development time.

Especially the last item is strongly related to the mapping task, which has to find a realizing component scenario for a selected service configuration. This requires an approach to evaluate the effectiveness of realizations in the sense of [16][17].

Acknowledgements This work is supported by the AmbiComp project, funded by the German Federal Ministry of Education and Research (BMBF – 01ISF05E). We would also like to thank our colleague Martin Becker for reviewing this paper.

10. REFERENCES