# Giving Users the Steering Wheel for Guiding Resource-Adaptive Systems

João P. Sousa, Rajesh K. Balan, Vahe Poladian,
David Garlan, Mahadev Satyanarayanan

December 2005
CMU-CS-05-198

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

This material is based upon work supported by the National Science Foundation (NSF) under Grant CCR-0205266, and by DARPA under Grant N66001-99-2-8918. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, DARPA, or Carnegie Mellon University.

**Keywords**: resource-aware systems, resource-adaptive applications, engineering adaptive systems, utility-based adaptation, adaptation policies, modeling user preferences, task-oriented computing, user studies, ubiquitous computing.

## ABSTRACT

Addressing resource variation plays an increasingly important role in engineering today's software systems. Research in resource-adaptive applications takes an important step towards addressing this problem. However, existing solutions stop short of addressing the fact that different user tasks often have specific goals of quality of service, and that such goals often entail multiple aspects of quality of service.

This paper presents a framework for engineering software systems capable of adapting to resource variations in ways that are specific to the quality goals of each user task. For that, users are empowered to specify their task-specific preferences with respect to multiple aspects of quality of service. Such preferences are then exploited to both coordinate resource usage across the applications supporting the task, and to dynamically control the resource adaptation polices of those applications. A user study validates that non-expert users can use our framework to successfully control the behavior of such adaptive systems.

# Contents

**1. INTRODUCTION ..... 1**

**2. FRAMEWORK ..... 2**

**3. USER PREFERENCES ..... 3**

3.1 Implementation ..... 5

3.2 Validation ..... 6

**4. COORDINATING RESOURCE USAGE ..... 9**

4.1 Implementation ..... 10

4.2 Validation ..... 10

**5. ADAPTIVE APPLICATIONS ..... 11**

5.1 Implementation ..... 12

5.2 Validation ..... 12

**6. RELATED WORK ..... 13**

**7. CONCLUSION ..... 14**

**8. REFERENCES ..... 15**

## 1. INTRODUCTION

Addressing resource variation is increasingly important in engineering computer systems. The reasons for that are twofold: first, resource availability often has a direct impact on the quality of service (QoS). For example, the frame rate of a video streamed over a network link may vary widely as a consequence of bandwidth fluctuations. Furthermore, unstable and limited resources may lead to undesirable and potentially disruptive application behavior: timeouts, faults, etc.

Second, software exposure to unstable resources is getting more common. In the growing domains of mobile and pervasive computing, it has been well documented that resources are fickle (e.g. [10,15]). Even applications originally designed for the relatively stable environment of desktops are increasingly being deployed on laptops equipped with wireless technology, and therefore faced with fluctuating bandwidth and limited battery charge.

One step towards addressing resource variation is building adaptive applications. Such applications monitor resource availability and adapt their computing tactics in order to maximize the QoS. Over the past few years, the community has developed a good understanding of resource-adaptation mechanisms (e.g. [8,9,22]).

Commonly, existing adaptive applications either enforce predetermined policies, or offer limited mechanisms to control those policies. In some cases, the adaptation mechanisms focus strictly on network conditions, enforcing policies that are established by system designers before the system is deployed. In other cases, users are offered limited control over the policies, typically focusing on a single aspect of QoS, such as battery duration.

Unfortunately, those limitations prevent adaptive systems from addressing two important goals. First, the optimal *resource allocation* among the applications being used should be determined by task-specific goals. For example, for a task such as preparing a presentation, the user may edit slides, browse the web for papers on the topic, and ask a colleague to email related material. Now, if the web browser is competing with email for limited bandwidth over a wireless link, how should bandwidth be allocated among the two? The answer depends on the user preferences for QoS aspects such as the response time of each application.

Second, the optimal *adaptation policies* should be determined by task-specific goals, and such goals often entail multiple aspects of QoS. For example, in the presence of limited bandwidth, should a web browser skip loading pictures in order to provide faster load times? For browsing citations, a user may prefer dropping images to improve load times; but for browsing online driving directions, the user may be willing to wait longer for the full page content.

This paper presents a framework for engineering resource-adaptive systems that (a) coordinate resource usage among several applications, and (b) enforce adaptation policies that take into account multiple aspects of QoS, and that may change dynamically according to the user preferences for each task.

To develop such a framework, important questions need to be answered. How to represent user preferences in a way that can be used to guide resource-adaptation policies? How to elicit such preferences? How to allocate resources among applications, and how to coordinate their adaptation policies? What APIs must applications expose to be amenable to such coordination?

To answer these questions, this work integrates results from three areas of research: eliciting user preferences, computing resource allocation dynamically based on user preferences, and enforcing resource usage and adaptation policies at the application level.

The contributions of this paper are: (1) a way to represent task-specific user preferences with respect to multiple aspects of QoS; (2) interfaces to elicit such preferences; and (3) an architecture for building resource-adaptive systems that coordinate resource usage across different applications, and that enable applications to dynamically adjust their policies based on user pref-

erences. This architecture is part of Project Aura [13], a wider project in ubiquitous computing. Another paper submission addresses a complementary but separate result: exploiting models of user tasks to support user mobility by automatically suspending and resuming tasks at different locations [26].

In the remainder of the paper, Section 2 discusses the proposed framework for engineering resource-adaptive systems, and specifically it describes the relevant slice of the Aura architecture. Sections 3, 4 and 5 elaborate respectively on the representation of user preferences, and on the aspects of the architecture related to coordinating resource usage and to application-level adaptation. Each of these sections includes a validation of the corresponding piece of research. Section 6 discusses related research. Section 7 puts this work into perspective, summarizing the main software engineering benefits it delivers, and pointing at future work.

## 2. FRAMEWORK

To address the fact that users often have specific QoS goals for different tasks, the framework for adaptation presented herein empowers users to specify their task-specific preferences with respect to multiple aspects of QoS. For that, however, we need to devise a representation of user preferences that is both practical from the point of view of elicitation from users, and powerful from the point of view of controlling adaptation.

Furthermore, we need to design an architecture that: (a) captures task-specific user preferences with respect to QoS, (b) coordinates the resource usage across the several applications supporting a user's task, and (c) enables applications to dynamically adjust their adaptation policies based on user preferences.

The architecture that supports this framework for adaptation is a slice of a broader architecture for ubiquitous computing developed in Project Aura (see Sousa'05 for more information [25]).

Figure 1 shows an informal view of the relevant components of Aura and their interactions: *Prism* captures user preferences; the *Environment Manager* (EM) coordinates resource usage across applications; and the *connectors* (arrows in black in the figure) disseminate user preferences and resource constraints as needed. *Chroma* is a software layer that facilitates the development of adaptive applications by providing common functionality to monitor and adapt to resource variation.

Aura supports a high-level notion of user tasks, such as preparing presentations or writing papers. Such tasks often involve several services. For instance, for preparing a presentation, a user may edit slides, refer to a couple of papers on the topic, check previous related presentations, and browse the web for new developments.

Whenever a user signals that he whishes to start or resume working on a task, Aura automatically identifies which applications available in the *environment*[1] are best to support all the required services, and activates them with the optimal resource allocation.

To achieve that, Prism communicates a task model to the EM. This model enumerates the required services and includes the user preferences for the task represented in the form of utility functions (more on this in Section 3).

Based on that model, the EM determines the best achievable *configuration*: a selection of applications to supply the services, together with the optimal allocation of resources among those applications (more on this in Section 4).

[1] We call computing *environment* to the set of devices, software components and resources accessible to the user at some point.

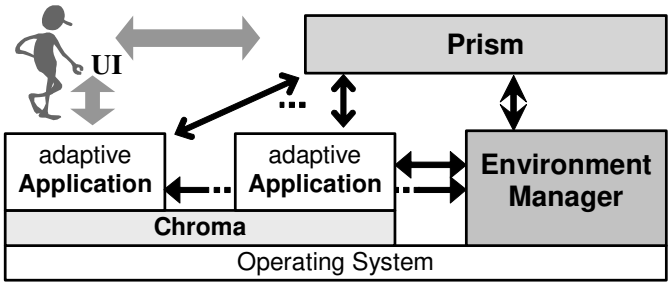



**Figure 1. Informal view of the components and interactions.**

Once Prism and the EM commit to a configuration, the EM communicates with the applications to activate the services and to establish the resource allocation. For example, if web browsing and watching a video stream are both part of the task, and competing for bandwidth, the utility for the user may be maximized when video streaming consistently uses 80% of the bandwidth and web browsing does not attempt to go beyond the remaining 20%.

Following the activation of services, Prism disseminates the user preferences for each service among the corresponding applications. Prism also disseminates user preferences while a task is being carried out: that occurs every time the user makes changes and hits *save* in the elicitation interfaces. In this case, the EM is also included in the broadcast, since the new preferences may imply a different allocation of resources.

Each adaptive application takes user preferences in the form discussed in Section 3 and translates them into concrete resource-adaptation policies. Specifically, each application chooses a computation tactic that optimizes the utility function passed by Prism, given the resource constraints established by the EM, and the available resources at each moment (more on this in Section 5).

The logical separation between Prism and the EM corresponds to the separation of what concerns users (service needs and QoS preferences for their tasks), and what concerns the environment: the availability of resources and their optimal allocation, given a set of requirements expressed as a task model.

Chroma is also concerned with the environment, but while the EM focuses in *overall* optimization, Chroma focuses on resource usage and adaptation *within* each application. Chroma provides mechanisms to (a) monitor available resources, (b) profile the resource demands of alternative computation tactics, and (c) decide dynamically which tactics to use depending on the available resources. Such decisions are taken at the granularity of small units of work, such as recognizing a speech utterance, or rendering one virtual-reality frame.

The connectors between Prism and the EM, between Prism and the applications, and between the EM and the applications, support the asynchronous exchange of XML messages over TCP. The asynchronicity of message exchange supports peer-to-peer protocols where every component may initiate communication, as needed, rather than standing on a purely reactive mode. The fact that messages are encoded in XML, as opposed to binary formats, facilitates implementing each component in the architecture using the language most convenient for the specific job.

## 3. USER PREFERENCES

The key requirements for Prism within the architecture outlined in Section 2 are: (a) to elicit preferences from non-expert users; and (b) to disseminate such preferences in a form that can be exploited to coordinate resource allocation and adaptation policies.

The models of user preferences in this context must take into account that: (1) users care about multiple aspects (dimensions) of QoS; (2) different services may be characterized by different

QoS dimensions; and (3) user preferences are task-specific. For example, for web browsing, users may care about load times and whether the full content is loaded (e.g., pictures); for automatic translation, users may care about the response time and accuracy of translation; for watching a movie, users may care about the frame update rate and image quality. From a user's point of view, the availability of resources, such as bandwidth, is important only to the extent that it impacts the observable QoS dimensions that users really care about.

A simple approach to modeling user preferences is to indicate which QoS dimension a user values the most. For example, for an automatic translation service, a user might indicate that response time is preferred over accuracy. The system could then adopt a policy that optimizes response time. However, important questions cannot be answered with this approach: for instance, how short of a response time will satiate the user? And even if accuracy is less important, what if it degrades so much that the translations become unusable?

At the other end of the spectrum, preferences may be expressed as an arbitrary function between the multivariate quality space and a utility space representing user happiness. For instance, the user might indicate that he would be happy with medium translation accuracy, as long as latency remains under 1 second, and that he will be happy to wait 5 seconds for highly accurate translations. Although fully expressive, designing mechanisms to elicit this form of preferences from non-expert users is a hard problem, and even more so if more than two QoS dimensions are involved.

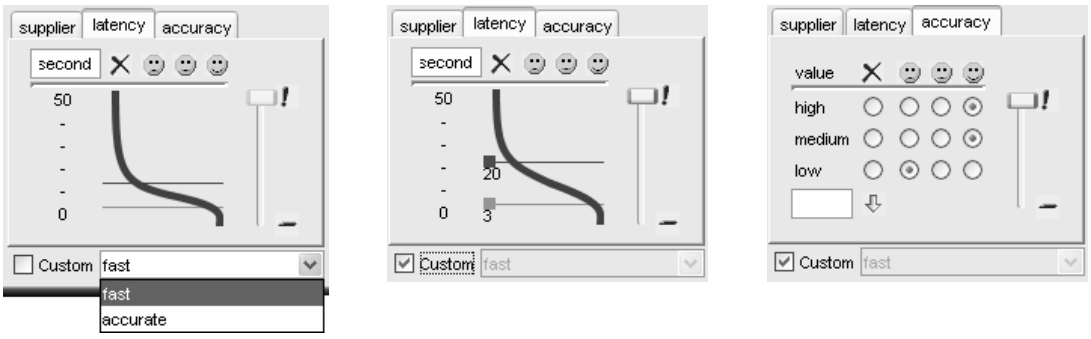
The approach we adopt sits between these two extremes. First, user preferences are expressed as independent utility functions for each QoS dimension. Such functions map the possible quality levels in the dimension to a normalized utility space $U \cong [0,1]$, where the user is happy with utility values close to 1, and unhappy with utility values close to zero. The functions for each dimension are then combined by multiplication, which corresponds to an *and* semantics: a user will be happy with the overall result only if he is happy with the quality along each and every dimension. Whenever a user task involves more than one service, the overall utility incorporates the QoS dimensions for all the services.

Second, for each continuous QoS dimension the user indicates two values: the threshold of satiation, and the threshold of starvation. For instance, the user might indicate that he would be happy with response times anywhere under 1 second, but would not accept response times over 10 seconds. Currently we use sigmoid functions to smoothly interpolate between these two zones: the threshold values corresponding to the knees of the curve (see Figure 2b. Preferences for discrete QoS dimensions, for instance *accuracy* encoded with values *high*, *medium* and *low*, are simply represented using a discrete mapping (table) to the utility space.

For evaluating an approach to modeling preferences, three criteria need to be considered: the expressiveness of the models, the ease of eliciting them, and the ease of exploiting them for adaptation.

Although mathematically less expressive than an arbitrary multivariate function, the user study described in Section 3.2, and our experience with multiple examples, confirm that our approach is both expressive enough for practical situations and accessible to non-expert users. Additionally, the algorithms for finding the optimal system configuration can be made simpler and more efficient by having a separate utility function for each QoS dimension as opposed to supporting arbitrary multivariate functions [23].

After deciding on *what* to model, the next question is *how* to represent those models. Such representations need to be passed to the adaptive applications at runtime. Specifically, every time the user changes his preferences, for instance due to starting a new task, the corresponding utility function needs to be passed to the application (or applications) serving the user, so that it can enforce the appropriate adaptation policies.




(a) (b) (c)

**Figure 2. QoS preferences for the language *translation* service.**

The representation of utility functions often used in isolated adaptive applications is a function, in the programming languages sense, that returns the utility value, i.e. the user happiness, given specific quality levels as parameters. However, even with dynamic load library (DLL) capabilities, such solution becomes cumbersome when different applications are written in different programming languages.

XML provides a more convenient representation of user preferences. Since XML is text-based, as opposed to binary, it can easily be passed among components at runtime, even when those components are distributed across the network (using protocols such as TCP or HTTP), even when those components reside in platforms with different binary encodings (e.g., Little Endian vs. Big Endian byte orderings [7]), and even when applications are written in different languages, since virtually all modern languages can access libraries for parsing and shipping XML.

## 3.1 Implementation

An important principle for interactive systems is that of offering incremental benefit for incremental effort (sometimes called *gentle slope systems* [18]). Users should able to reap significant benefits from adaptation even with little effort put into describing their preferences, and the more effort users are willing to put, the better job the system can do in following their preferences.

Prism offers predefined templates associated with each service type as a quick way for defining user preferences. At least one template, the default, is provided, and depending on the service type, more templates may be available. For example, for web browsing, the *default* template specifies that pictures should be loaded but is lenient on the response times. An additional *fast* template is defined, specifying stricter constraints for the response time but being tolerant of not loading pictures.

Whenever the preferences specified by the default template fail to capture the user's intent for the service within a particular task, the user may select another template for the service. If the user is not satisfied by the available templates, he may customize the QoS preferences for the service.

Figure 2 illustrates the user interfaces Prism provides for eliciting QoS preferences. The example shown is for the service of translating natural language, which has two QoS dimensions: *latency* (response time) and *accuracy*.[2] The latency of recognizing a sentence has a numeric domain and is expressed in seconds. The accuracy of translation reflects how well the meaning is preserved.

[2] Prism also allows users to specify which applications are preferred to *supply* each service (leftmost tabs in Figure 2). This is part of a broader framework for adaptation [25].

```
<utility combine="product">
  <QoSdimension name="latency" type="int">
    <function type="sigmoid" weight="1">
      <thresholds good="3" bad="20" unit="second"/>
    </function>
  </QoSdimension>
  <QoSdimension name="accuracy" type="enum">
    <function type="table" weight="1">
      <entry x="high" f_x="1"/>
      <entry x="medium" f_x="1"/>
      <entry x="low" f_x="0.3"/>
    </function>
  </QoSdimension>
</utility>
```

**Figure 3. Internal representation of the preferences in Figure 2**

Two predefined templates are available: the *fast* template has stricter constraints on latency than the *accurate* template, but tolerates a lower accuracy of translation. When a user starts a language translation service, the *fast* template is activated by default. Users may switch between these two templates in the selection box shown at the bottom-right in Figure 2 (a).

The utility function for latency is a sigmoid function where the thresholds of satiation and starvation are represented by the green (lighter) and red (darker) lines in Figure 2 (b), respectively. The user happiness corresponding to each value of latency is indicated by the scale at the top, ranging from a happy face (☺) for values beyond the satiation threshold, all the way down to a cross (✕), representing rejection, for values beyond the starvation threshold. The utility function for accuracy, represented in Figure 2 (c), is a table indicating the user happiness for each value of accuracy.

In addition to showing the precise meaning of preference templates, these interfaces support customizing the user preferences for a specific task. After selecting the custom check-box to the left of the template selection box, users may depart from a template and set their preferences directly. The thresholds of sigmoid curves may be adjusted by dragging the green and red handles, respectively. The entries in tables may be changed by selecting the utility for each value in the QoS dimension.

Figure 3 shows the XML representation of the preferences captured in Figure 2. Prism creates user interfaces like the one in Figure 2 dynamically, based on the specific QoS dimensions present in the XML representation. In turn, the XML representation is updated by manipulating the widgets in the user interface. Whenever the user commits to (saves) a change in his preferences, Prism transmits an XML structure like the one in Figure 3 to the application supplying the corresponding service.

## 3.2 Validation

Validating the proposed solution entails demonstrating (a) that non-expert users can interact successfully with Prism to satisfy their QoS goals; and (b) that Prism can coordinate the policies of adaptive applications using such a representation of user preferences. To better control the conditions of the evaluation, we performed a separate experiment for each of these premises employing a *Wizard of Oz* approach.[3]

First, a user study investigated whether non-expert users can use Prism to control the adaptation policies enforced by an interactive application. The experiment involved using an automatic natural language translator running on a mobile device. The quality of translation would vary, since the translator would run sophisticated algorithms on a remote server, in the presence of fluctuating bandwidth and available capacity in the server. To prevent limitations in the capa-

[3] In a Wizard of Oz experiment, a simulation substitutes for part of the system, in this case to facilitate exercising the other parts in specific ways that would be otherwise hard to achieve.

bilities of the actual translation application (limited dictionaries, etc.) from affecting the participants' perception of the accuracy of translation, we replaced a human for the remote translation server. The role of this person, part of the research team, was not revealed to the participants during the study.

Second, system testing investigated the effectiveness of controlling adaptation policies using a wide range of realistic user preferences. Since covering a large number of preferences under different resource conditions would be tedious and cumbersome to carry out with real users, we substituted a batch simulator for the user, which bypassed the user interfaces of both Prism (Figure 2) and the automatic translator.

The remainder of this section elaborates on the user study and the corresponding results, whereas the system testing is discussed in Section 5.2.

The user study focused on answering the following questions: first, can users recognize the applicability and use preference templates to achieve a goal? Second, can users think of and manipulate preferences in terms of thresholds? Third, do they find it easy? And fourth, can users interpret the effects of specifying different preferences in the application's adaptive behavior?

The participants in the user study were drawn from a population with homogeneous education level and age group, but diverse technical background. Ten students in the age group 18-29 were drawn among the respondents to a posting, 5 of which from computing-related fields (computer science, electrical and computer engineering, logic) and the remaining 5 from other fields (business, physics, literature). Incidentally, 6 were male and 4 female.

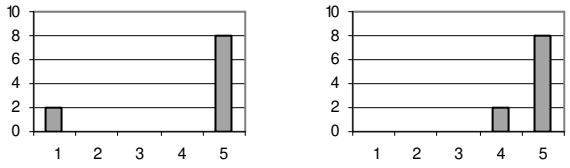
Participants individually performed an experiment that lasted 30 minutes, after being given a 20 minute introduction that included a description of the scenario, an introduction to the *think aloud* protocol [27], and hands-on practice with the adaptive translator. During the experiment, all of the participant's actions on the screen, in addition to their voice were recorded using video capturing software [28]. After the experiment, the participants completed a short questionnaire.

The scenario for the experiment revolved around a conversation with a foreign language speaker (Spanish in this case) aided by translation software. To prevent serious misunderstandings in a real situation, users of the translation software would be able to check the accuracy of translation by having the Spanish translation translated back to English and spoken (using speech synthesis) on the user's earphones. Users would press a go-ahead button to synthesize the Spanish translation only if they were happy with the accuracy of translation.

During the experiment, participants were asked to input sentences of their own making, listen to the output of the double translation, and rate the accuracy. The training included calibrating the participants' rating of accuracy using the following scale: high, if the meaning is fully preserved; medium, if the meaning is roughly preserved; and low, if the meaning is seriously distorted.

Participants were asked to pursue different QoS goals during each part of a three-part experiment. Within each part, we simulated resource variation and asked the participants to evaluate the changes both in latency and accuracy of translation. During the first two parts, the QoS goals could be satisfied by preference templates. During the third part, the specific goal could only be achieved by customized preferences. The participants were not directed as to whether or not to use templates in any case.

Whenever the QoS goals could be met by a template, the participants did use templates in 17 out of 20 cases. In the remaining 3 instances, the participants were still able to achieve the goals using customized preferences. When asked about the clarity and usefulness of templates, 8 participants were fully favorable, while 2 didn't recognize a benefit in having templates (Figure 4a).




**(a) Templates** **(b) Thresholds**

**Figure 4. Likert scale evaluation of preferences' specification (5-fully favorable, to 1-unfavorable).**

All 10 participants were able to manipulate the thresholds in customized preferences for achieving the required QoS goals. Specifically, the experiment was set in such a way that the thresholds in one dimension needed to be made stricter, while relaxing the other dimension, under penalty of the goal not being achievable.

When asked about the clarity of using thresholds to specify preferences, 8 participants were fully favorable, while 2 thought some alternative strategy could be preferable (Figure 4b). One of these participants suggested that an X-Y representation of the actual tradeoff of accuracy versus latency would make it clear what is the impact on one caused by that pushing the other. However, there are two reasons why such representation may not be such a good idea. First, such representation exposes users to details they should not have to worry about: the specific tradeoff between dimensions is a function of the available resources, which changes from one moment to the next. In contrast, the current representation focuses exclusively on the thresholds of user happiness along each dimension, and as such, is independent of resource fluctuations. Second, for all practical purposes an X-Y plot is limited to tradeoffs between two variables: it would be very hard to manage graphical representations with more than two dimensions.

When asked about how easy it was to use the interfaces in Figure 2 to customize preferences, 5 participants were fully favorable while the other 5 thought the interfaces could be improved. Nevertheless, the time it took to customize preferences was mostly under 1 minute, the average being 42 seconds. To provide a comparison with these results, one of the authors measured the time it took to customize a power-saving profile on an IBM ThinkPad T30 computer using the IBM Battery Maximizer program. Creating a new profile by specifying a subset of all possible settings on a single window took 25 seconds. Modifying an existing profile by using a wizard that allows setting more options took 45 seconds. We should note that this particular author had used the program several times before. Thus, we conclude that the time it took the study participants to customize preferences is in line with performing similar tasks on a mobile computer.

The participants were able to interpret the effects of specifying different preferences in the application's adaptive behavior. To verify this, we tested the hypothesis that when resources change participants perceive a change in the QoS, with a greater impact along the QoS dimension for which the preferences are laxer. For that, after each translation the participants evaluated which QoS dimension changed the most relative to the previous translation: a noticeable change in accuracy with similar latencies, a noticeable change in latency with similar accuracies, no noticeable changes, etc. Participants then related those changes with the strictness or laxness of the preferences along each QoS dimension. The participants were not informed of when or in which direction resources would change.

Figure 5 shows the results of correlating which dimension had stricter preferences with which dimension was perceived to have changed the most. The correlation coefficient is negative, meaning that whenever user preferences were stricter along one dimension, the participants perceived a greater fluctuation on the other dimension (caused by underlying resource fluctuations).

| Correlation Coefficient | t-value | Significant at 95% |
|---|---|---|
| -0.6 | -4.27 | Yes |

**How to interpret a correlation**: the correlation coefficient denotes the slope of the line that best fits the data. A positive coefficient means that an increase in the x-axis corresponds to an increase in the y-axis. A negative coefficient means that an increase in the x-axis corresponds to a decrease in the y-axis. If the coefficient is zero, the data cannot be approximated by a straight line (there is no correlation between the x values and the y values).

**Student's t-test of significance**: indicates the likelihood that the correlation in the data sample corresponds to a real correlation in the general population. A commonly accepted threshold is 95% confidence. Statistics manuals contain tables of t-statistics for each size of the data sample. The t-test consists of comparing the t-value calculated for the correlation with the lookup t-statistic. If the absolute t-value is larger than the t-statistic, then the correlation is significant with 95% certainty.

**Sample**: 40 data points relating two variables (38 degrees of freedom), for which the t-statistic is 2.024 for a 95% confidence.

**Figure 5. Regression performed on experiment data.**

This user study demonstrates that non-expert users can both define their preferences using Prism, and interpret the results of such definitions in the system's QoS. A control loop is therefore closed, enabling users to pursue concrete QoS goals. The practicality of the control loop is confirmed by the fact that all participants were always able to achieve concrete QoS goals, within reasonable times.

## 4. COORDINATING RESOURCE USAGE

The key requirements for the Environment Manager (EM) within the architecture outlined in Section 2 are: (a) select the set of applications that best serves the user's needs and preferences for a task; (b) coordinate the resource usage of that set of applications; and (c) perform these two roles without introducing perceptible delays for the user, and without drawing significantly from the resources available to the applications.

Furthermore, these requirements need to be satisfied on an ongoing basis, adjusting the allocation of resources or even the set of applications in response to changes on either the user's task, or on the environment (e.g., faults or significant changes in resources).

The EM takes three kinds of inputs:

**Task models**. A task model enumerates the service needs for the user's task, and the user preferences for each QoS dimension associated with those services. Such preferences are represented as a mathematical function, selected from a vocabulary of functions (sigmoids, tables…) shared by Prism, the EM and the adaptive applications. These functions are combined by multiplication into an overall utility function for the task [25].

**Capability profiles**. The capability profile of an application relates the possible quality levels that the application can provide with the corresponding resource demands. Unlike more fine-grained information maintained by adaptive applications, this is a summary view containing averages over a large number of historical samples [23].

**Resource estimates**. Resource estimates take a snapshot of the overall resources available in the environment. Similarly to capability profiles, this information is kept at a coarse level (several seconds), smoothing-out momentary drops and surges in resources. In contrast, adaptive applications focus precisely on those drops and surges to accomplish fine-grain adaptation [3].

Given the three inputs above, the EM computes an optimal *configuration*: (1) a suite of applications, and (2) a resource allocation for each selected application. The resulting configuration is optimal in the sense that it maximizes the overall utility function for the task. For the maximization, the EM employs a search algorithm that efficiently searches the complete configuration space of available applications and their quality levels, under the constraints of the available resources. Having found the solution, the EM invokes the selected applications to activate the services, and advises each application of its resource allocation.

To compute the optimal configuration, there are two alternative strategies: reactive, and anticipatory. The anticipatory strategy requires the same kinds of inputs, but those inputs are enriched with a temporal perspective.

In the reactive strategy, a snapshot of all the inputs is fed to an optimizer, which determines the optimal configuration. Then, periodically, as either the user's preferences or environment change, the optimizer re-computes the optimal configuration, possibly changing the suite of applications[4] or the allocation of resources among them.

In the anticipatory strategy of coordination, the optimizer takes not only the current snapshot of the inputs, but also their expected paths over time. Hence the decision is not "optimal now", but "optimal over time," given the expected changes over a future window. This strategy is object of ongoing research.

## 4.1 Implementation

The current implementation of the EM adopts a reactive strategy of coordination. The optimization algorithm takes advantage of two components of utility: one that rewards a particular choice of applications (based on user familiarity with those applications), and another that rewards the achievable QoS.

The outline of this algorithm is as follows (details in [23]):

- Query for the required services among all available applications.
- Generate all possible suites of applications that jointly satisfy the service needs of the task.
- For each suite, compute the utility component that reflects how much the user values the particular choice of applications (prior to considering the QoS they can deliver). Then sort the suites according to the resulting score.
- Starting from the highest scoring suite, search the joint quality space of the suite, under the current resource constraints, to find the point that delivers the maximum utility. This search problem is equivalent to a multiple-choice, multiple-dimension knapsack packing problem, which searches for a set of highest value items given multiple resource constraints.
- Break from the search as soon as none of the remainder candidate suites can beat the current maximum utility, no matter how good of a QoS they can provide. Pick the suite with the highest utility found so far, and allocate the resources corresponding to the point in the quality space that maximizes utility.

## 4.2 Validation

The requirements of selecting the set of applications that best serve the user's needs, and of coordinating resource usage among those applications, are addressed by the optimization algorithm outlined above. This algorithm is run in response to changes in user needs, and periodically to address trends in resource availability. This ensures that the optimality of configuration is maintained over time.

---

[4] Oscillating between closely rated configurations is prevented by introducing hysteresis based on change penalties.

To validate that running this algorithm does not introduce perceptible delays for the user, and does not draw significantly from the resources available to the applications, we performed a system's evaluation. The experiments were carried out on an IBM ThinkPad 30 laptop running Windows XP Professional, with 512 MB of RAM, 1.6 GHz CPU, and WaveLAN 802.11b card. Prism and the EM each run on a Hot Spot JRE from Sun Microsystems, version 1.4.0_03.

The average latency finding the optimal configuration is 200 ms (standard deviation 50 ms) for user tasks requiring from 1 to 4 services, when 4 to 24 alternative suites of application are available.[5] We also measured the overall overhead of activating all the services in a task, which includes the latency of the communication protocols in Figure 1 (but not the latency of starting the applications themselves). This latency is mostly constant, rather than growing proportionally to the number of services, and is on average 700 ms (standard deviation 200 ms).

The memory footprint of the EM ranges linearly from 7 MB to 15 MB when it holds the descriptions of 20 up to 400 services in the environment. By comparison, a "hello world" Java application under the used Java Runtime Environment (JRE) has a memory footprint of 4.5 MB, and a Java/Swing application that shows a "hello world" dialog box has a memory footprint of 12 MB. When reevaluating the resource allocation every 5s, the EM uses on average 3% of CPU cycles.

In conclusion, the delay introduced by the infrastructure in Figure 1 is mostly imperceptible for the user when coupled with starting up applications, and the EM's footprint is comparable to the smallest applications.

## 5. ADAPTIVE APPLICATIONS

The key requirements for adaptive applications within the architecture outlined in Section 2 are: (a) to comply with the resource allocation determined by the EM, and (b) to optimize the utility functions passed by Prism, in the face of resource variations.

To satisfy these two requirements, applications use mechanisms to (a) monitor available resources, (b) profile the resource demands of alternative computation tactics, and (c) decide dynamically which tactics to use depending on the available resources.

To avoid the costs of replicating such functionality in every application, Aura provides Chroma, a common layer for resource adaptation [3]. To further assist adaptive applications running on small platforms, Chroma supports partitioning the application's logic into modules that can reside on remote servers.

For that, Chroma exploits the idea of *tactics*: descriptions of valid sequences of module invocations that accomplish some unit of work. At runtime, Chroma determines the appropriate tactic to use and where to run each module: either locally or at a specific remote server. Chroma selects the tactic that a) maximizes the utility function representing the current user preferences, and (b) does not exceed the resource constraints for the application. For executing a tactic, Chroma handles all necessary remote calls, argument passing, and intermediate buffer management. Applications just have to provide Chroma with the arguments for the tactic and Chroma will return the computed output.

Chroma takes four kinds of inputs:

**Tactics**: describe the alternative computation tactics for each application, including tactics that involve remote execution.

**Utility functions**: represent user preferences for each QoS aspect.

[5] This represents an improvement of about a factor of two over the numbers reported in [23].

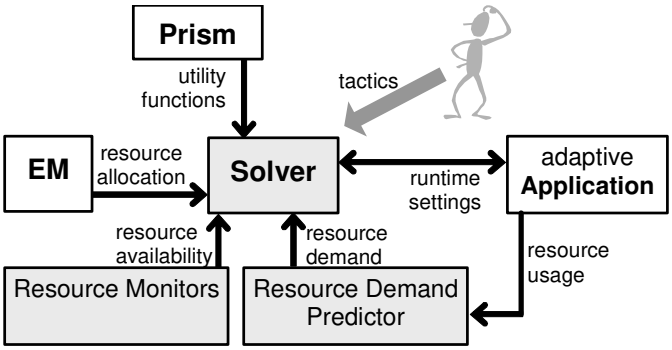



**Figure 6. Components and information flow in Chroma.**

**Resource allocation**: constrains the resource usage by each application, even if more resources are available. These bounds are in place to optimize overall QoS goals.

**Resource usage**: monitors the resources used by each tactic.

## 5.1 Implementation

The current implementation of Chroma is informally depicted in Figure 6. The main components of Chroma are highlighted: resource monitors, a resource demand predictor, and the solver, which determines the optimal application runtime settings.

Currently supported resource monitors include the available bandwidth, battery charge, CPU and memory, both on the local machine and on any remote servers. Chroma uses the history-based resource demand predictors developed by Narayanan et al [19]. The solver performs an exhaustive evaluation of all possible tactics and picks the one that is forecasted to make the best use of the available resources with respect to the utility functions.

To integrate an application with Chroma, a (human) system integrator needs to do two things: describe the application's tactics, and insert four Chroma API calls into the application's code. Two of these calls register and deregister the application with Chroma. The file containing the description of tactics is read upon registration.

The two other calls to Chroma are invoked when performing a unit of work, such as translating an utterance, or playing a video segment. First, the application calls Chroma to decide which tactic should be run and the corresponding runtime settings. Once informed of this decision, the application calls Chroma to execute the selected tactic (possibly involving remote execution) with the specific parameters (e.g., the utterance to be translated).

The process of integrating an application with Chroma is described in detail in Balan'05 [3].

## 5.2 Validation

The validation presented in this section corresponds to the system testing part of the experiment described in Section 3.2. A more thorough validation of Chroma's ability to perform adaptation in the presence of limited resources is presented in Balan'03 [2].

This test is based on the scenario where a PDA is used to carry out natural language translation. When resources are *poor,* no remote servers can be reached, and the translation is carried out exclusively using the PDA's capabilities. When resources are *rich,* powerful remote servers are available to do part of the work. We used a 233Mhz Pentium laptop with 64MB of RAM to simulate the PDA and 1GHz Pentium 3 laptops with 256MB of RAM as the remote servers.

The test used 3 randomly selected sentences, of between 10-12 words in length. Each sentence was (doubly) translated five times, from English to Spanish and then back into English using Pangloss-Lite, a language translation application [12].

**Table 1. Relative utility of Chroma's decisions**

| preferences / resources | Chroma's decision | Lower resources | Higher resources |
|---|---|---|---|
| *fast / poor* | 1.0 | N/A | 0.45 |
| *accurate / poor* | 1.0 | 0.37 | 0.13 |
| *fast / rich* | 1.0 | 0.51 | 0.83 |
| *accurate / rich* | 1.0 | 0.50 | N/A |

The utility functions provided to Chroma correspond to the *fast* and *accurate* templates introduced in Section 3.1. The fast template accepts medium accuracy within 1s, and the *accurate* template is willing to wait 5s for highly accurate translations.[6] Each sentence was translated under rich and poor resources, for each of the two preference templates (four test situations).

Table 1 shows the relative utility of Chroma's decisions in each of the four test situations. The relative utility is calculated as the utility of the QoS delivered by Chroma's decision relative to the best possible utility, among all the alternative tactics, given the current resource conditions. To illustrate this optimality, the two rightmost columns show the utility of the adjacent decisions in terms of resource usage. That is, the relative utility of the decisions that would take the nearest lower resources, and the nearest higher resources, respectively. There are two corner cases, shown with N/A, where Chroma's decision corresponds to the lowest possible resource usage, and to the highest possible usage.

In summary, Chroma always picks the best possible tactic, under different resource conditions, and different user preferences.

## 6. RELATED WORK

To the best of our knowledge, our work is the first that integrates task-specific user preferences into a complete resource adaptation solution. The presented framework ties together three major pieces: Prism to elicit task-specific user preferences; the Environment Manager to compute the optimal resource allocation among applications based user preferences; and Chroma to provide OS-level support to resource adaptation.

Prism uses theories and techniques from microeconomics to elicit utility with respect to multiple attributes. Recent results in software engineering have leveraged similar ideas. For example, in the Security Attribute Evaluation Method (SAEM), the aggregate threat index and the losses from successful attacks are computed using similar utility functions [5]. The Cost Benefit Analysis Method (CBAM) uses a multidimensional utility function with respect to quality attributes in evaluating multiple architectural alternatives for a software project [17]. Our work is different from SAEM and CBAM in that it is geared towards pervasive and mobile computing.

User studies done in mid-to-late 1990s have demonstrated that certain dimensions of quality of service (e.g., absence of jitter) are much more important to the user than improvements in frame rate [20]. Our system recognizes the importance of these results and ensures, by explicit resource arbitration, that adequate resources are available for applications to provide service while maximizing the task-level utility.

From analytical point of view, closest to the role of the Environment Manager are Q-RAM [16], Knapsack algorithms, and winner determination in combinatorial auctions. Our work goes a step further and also considers the problem of selecting applications and reducing unwanted disruption to the user. Work presented in [21] performs utility-based resource allocation based on congestion prices and shares the same objective, but does not utilize a centralized arbiter of re-

[6] The latency thresholds for the system testing are much smaller than the ones used in the user study in Section 3.2, in which the translation was performed by a team member.

sources. Relative to our solution, [21] has two shortcomings: (1) reaching an equilibrium state might take a long time, and (2) even after the equilibrium is reached, there is no guarantee that it is globally optimal.

From an analytical point of view, closest to the role of the Environment Manager are Q-RAM [16], Knapsack algorithms, and winner determination in combinatorial auctions. Our work goes a step further and also considers the problem of selecting applications and reducing unwanted disruption to the user. Neugebauer et al performed utility-based resource allocation based on congestion prices [16]. They shared the same objective, but did not utilize a centralized arbiter of resources. Relative to our solution, their work has two shortcomings: (1) reaching an equilibrium state might take a long time, and (2) even after the equilibrium is reached, there is no guarantee that it is globally optimal.

Dynamic resolution of resource allocation policy conflicts involving multiple mobile users is addressed in [6] using sealed bid auctions. While this work shares utility-theoretic concepts with our configuration mechanisms, the problem we solve is different. Our work has no game-theoretic aspects and addresses resource contention by multiple applications working for the same user.

Chroma uses adaptation techniques first proposed in Odyssey [22], and it incorporates the history-based resource prediction techniques developed by Narayanan et al [19]. Even though, the use of remote execution to overcome resource constraints has been previously explored [9,24], Chroma's use of tactics to determine optimal application partitioning is unique [3].

Finally, some previous work, such as the GRACE Operating System [29], has claimed to present a complete user-centric adaptation solution. Our framework goes a step further by addressing the elicitation of task preferences. Furthermore, the notion of task-level utility is much more fine-grained in our work.

## 7. CONCLUSION

Today's default architecture for adaptive systems places the responsibility of eliciting user preferences and managing resource usage on each application. Unfortunately this approach makes it very hard to grant users adequate control over the QoS they get in the presence of limited or fluctuating resources.

There are two main reasons for that shortcoming: first, it is costly to replicate mechanisms for eliciting user preferences and for resource adaptation across applications. Therefore, many applications implement limited solutions for adaptation, or none at all.

Second, it is hard to coordinate resource usage among the applications supporting the user's task. Each individual application has to determine its own optimal resource allocation so that the user's overall QoS goals for the task are satisfied. However determining that optimal point requires global knowledge about how the resource demand of every relevant application correlates with the desired QoS levels. Unfortunately, it is hard for an individual application to even know which other applications are involved in supporting the user's task. Consequently, today individual applications trample on each other in their quest for resources, oblivious of the user's overall QoS goals.

This paper presents an architecture that addresses these difficulties by factoring common adaptation concerns into a common infrastructure which then coordinates resource usage and adaptation polices. This work integrates results from three areas of research: (1) eliciting user preferences, (2) computing resource allocation dynamically based on user preferences, and (3) enforcing resource usage and adaptation policies at the application level.

The architecture presented herein delivers a number of benefits for engineering adaptive applications.

First, it introduces a way to represent user preferences that is both practical from the point of view of elicitation, and powerful from the point of view of guiding adaptive applications. As documented in Section 3.2, non-expert users can successfully interact with Aura to specify their preferences and pursue concrete QoS goals. As documented in Sections 4.2 and 5.2, Aura can exploit that representation to dynamically coordinate the resource usage and the adaptation policies of individual applications.

Second, it introduces a component whose primary responsibility is to elicit user preferences and track changes in those preferences. This both reduces the cost of implementing similar mechanisms in every application, and allows for the creation of preferences that are user-specific, and furthermore, task-specific, rather than application-specific. Today, users increasingly carry out their tasks across different devices, using different applications, and the same application may be used to carry out tasks with very distinct QoS goals. In this reality, it becomes easier to decide which user preferences to apply at each moment from within a component that takes the perspective of users and user tasks, than from within an individual application.

Third, it introduces a component whose primary responsibility is to coordinate resource usage among all the applications supporting the user's task. The system-wide perspective held by this component makes it feasible to find the resource allocation that optimizes overall QoS goals, rather than local ones. By providing this component with the user preferences for each task, the resource allocation is made to match the task's specific QoS goals.

Forth, it introduces a component that facilitates extending applications for adaptation. This component saves development costs by providing common mechanisms for (a) monitoring available resources, (b) profiling the resource demands of alternative computation tactics, and (c) deciding dynamically which tactics to use depending on the available resources. The efficiency of these mechanisms makes it possible to react to resource variations at a time granularity of milliseconds.

Future work will focus on achieving an even more effective resource usage by factoring in knowledge about alternative and future needs of the user, and about resource availability in the near future and in the vicinity of the user's current location.

Specifically, we hypothesize that the optimization algorithms can be extended to run over an interval of time, factoring in forecasts of resource availability and of user QoS goals. Furthermore, rather than offering marginally satisfactory, even if "optimal," QoS for a task in the presence of limited resources, Aura could recognize that another task that the user might also be willing to work on is a better match for the available resources. Finally, Aura could recognize that the particular task the user is engaged on would be much better supported at the room down the hall.

## 8. REFERENCES


1. Amiri, K., Petrou, D., Ganger, G., Gibson, G. Dynamic function placement for data-intensive cluster computing. *Procs USENIX 2000 Annual Technical Conf*, San Diego, CA, 2000.
2. Balan, R.K., Satyanarayanan, M., Park, S., Okoshi, T. Tactics-Based Remote Execution for Mobile Computing. *Procs 1st USENIX International Conference on Mobile Systems, Applications, and Services* (MobiSys), San Francisco, CA, 2003.
3. Balan R.K., Gergle, D, Satyanarayanan, M, Herbsleb, J., Simplifying Cyber Foraging for Mobile Devices, *Carnegie Mellon University Technical Report CMU-CS-05-157R*, 2005. Also submitted for publication.
4. Basney, J. and Livny, M. Improving Goodput by Coscheduling CPU and Network Capacity. *Intl. Journal of High Performance Computing Applications*, 13(3), Fall 1999.

5. Butler, S. Security Attribute Evaluation Method. A Cost-Benefit Approach. *Proc Int'l Conf in Software Engineering (ICSE)*, Orlando, FL, 2002.
6. Capra, L., Emmerich, W., Mascolo, C. CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10), pp 929-945, 2003.
7. Cohen, D. On Holy Wars and a Plea for Peace. *Computer* 14:10, pp 48-54, October 1981.
8. De Lara, E., Wallach, D., Zwaenepoel, W. Puppeteer: Component-based Adaptation for Mobile Computing. *Proc. 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.
9. Flinn, J., Satyanarayanan, M. Energy-aware Adaptation for Mobile Applications. *Procs 17th ACM Symposium on Operating Systems Principles* (*SOSP*), pp 48-63, Kiawah Island, SC, 1999.
10. Forman, G., Zahorjan, J. Survey: the Challenges of Mobile Computing, *IEEE Computer*, 27(4), pp 38-47, 1994.
11. Fox, A., Gribble, S., Brewer, E., Amir, E. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. *Procs 7th Intl Conf on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), ACM Press, pp 160-170, Cambridge MA, 1996.
12. Frederking, R., Brown, R. The Pangloss-Lite Machine Translation System. *Expanding MT Horizons: Procs 2nd Conf Association for Machine Translation in the Americas*, pp 268-272, Montreal, Canada, 1996.
13. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste P. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, April-June 2002.
14. Hunt, G. C. and Scott, M. L. The Coign automatic distributed partitioning system. *Proc. 3rd Symposium on Operating System Design and Implementation (OSDI)*, New Orleans, LA, Feb. 1999.
15. Katz, R. Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Comms*, 1(1), pp 6-17, 1994.
16. Lee, C. et al. A Scalable Solution to the Multi-Resource QoS Problem. *Proc IEEE Real-Time Systems Symp (RTSS)*, 1999.
17. Moore, M.; Kazman, R.; Klein, M.; & Asundi, J. Quantifying the Value of Architecture Design Decisions: Lessons from the Field, *Procs 25th Int'l Conf on Software Engineering (ICSE)*, Portland, OR, 2003.
18. Myers, B., Smith, D., Horn, B. Report on the 'End-User Programming' Working Group. *Languages for Developing User Interfaces*. Jones and Barlet (Eds.), Boston, MA, 1992.
19. Narayanan, D., Flinn, J., Satyanarayanan, M. Using History to Improve Mobile Application Adaptation. *Procs 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Monteray, CA, 2000.
20. Nerode, A., Parikh, S., Srivastava, S., Varadarajan, S., Wijesekera. QoS based Evaluation of the Berkeley Continuous Media Toolkit, *University of California at Berkeley Technical Report A461563*, March 1, 1999.
21. Neugebauer, R., McAuley, D. Congestion Prices as Feedback Signals: An Approach to QoS Management. *Procs ACM SIGOPS European Workshop*, Kolding, Denmark, 2000.

22. Noble, B., Satyanarayanan, M., Narayanan, D., Tilton, E.J., Flinn, J., Walker, K.R. Agile Application-Aware Adaptation for Mobility. *Procs 16th ACM Symposium on Operating Systems Principles* (*SOSP*), Saint-Malo, France,1997.

23. Poladian, V., Sousa, J.P., Garlan, D., Shaw, M. Dynamic Configuration of Resource-Aware Services. *Procs 26th Intl Conf on Software Engineering - ICSE 2004*, IEEE Computer Society, pp. 604-613, Edinburgh, UK, May 2004.

24. Rudenko, A., Reiher, P., Popek, G.J., Kuenning, G.H. Saving Portable Computer Battery Power Through Remote Process Execution, *Mobile Computing and Communications Review* (*MC2R*), 2(1), pp 19-26, 1998.

25. Sousa, J.P. Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments. PhD thesis available as *Carnegie Mellon Univ. Technical Report CMU-CS-05-123*, 2005.

26. Sousa, J.P., Garlan, D. A New Approach to Supporting Mobile Users: Task-Model Mobility, *submitted for publication*.

27. Steinberg, E., editor. *Plain language: Principles and Practice* Wayne State University Press, 1991.

28. TechSmith Corporation. Camtasia Studio. www.techsmith.com, June 2004.

29. Yuan, W., Nahrstedt, K., Adve, S,. Jones, D. and Kravets, R. GRACE: Cross-Layer Adaptation for Multimedia Quality and Battery Energy. To appear in *IEEE Transactions on Mobile Computing*.