A Software Infrastructure for User–Guided Quality–of–Service Tradeoffs

João Pedro Sousa[1], Rajesh Krishna Balan[2], Vahe Poladian[3], David Garlan[3], and Mahadev Satyanarayanan[3]

[1] Computer Science Department, George Mason University
4400 University Drive, Fairfax VA, USA
[2] School of information Systems, Singapore Management University
80 Stamford Road, Singapore
[3] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA, USA
jpsousa@cs.gmu.edu, rajesh@smu.edu.sg
{poladian,garlan,satya}@cs.cmu.edu

Abstract. This paper presents a framework for engineering resource-adaptive software targeted at small mobile devices. Rather than building a solution from scratch, we extend and integrate existing work on software infrastructures for ubiquitous computing, and on resource-adaptive applications.

This paper addresses two research questions: first, is it feasibility to coordinate resource allocation and adaptation policies among several applications in a way that is both effective and efficient. And second, can end-users understand and control such adaptive behaviors dynamically, depending on user-defined goals for each activity. The evaluation covered both the systems and the usability perspectives, the latter by means of a user study.

The contributions of this work are: first, a set of design guidelines, including APIs for integrating new applications; second, a concrete infrastructure that implements the guidelines. And third, a way to model quality of service tradeoffs based on utility theory, which our research indicates end-users with diverse backgrounds are able to leverage for guiding the adaptive behaviors towards activity-specific quality goals.

Keywords: Mobile computing, Resource adaptation, Self-adaptive systems, Software architecture, User studies.

1 Introduction

Sophisticated software is increasingly being deployed on small mobile devices, taking advantage of their growing capabilities and popularity. Media streaming is already found frequently in PDAs and high-end cell phones. Soon, applications such as speech recognition, natural language translation, and virtual/augmented reality may leap from research prototypes to widespread commercial use.

While software has enjoyed plentiful and stable resources in the world of desktops (and to some extent, of laptops,) resource variation needs to be taken into account in smaller devices. Despite the impressive capabilities of today's mobile devices, user

J. Cordeiro et al. (Eds.): ICSOFT 2008, CCIS 47, pp. 48–61, 2009.

expectations with respect to performance and sophistication will continue to be set by the full-size versions running on powerful desktops and servers.

Research in resource-adaptive applications takes an important step towards addressing resource limitation and variation [1-3].

However, existing solutions either enforce predetermined policies, or offer limited mechanisms to control the application's 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 quality of service, such as battery duration.

Unfortunately, those limitations prevent adaptive systems from addressing two important issues. First, user goals often entail *tradeoffs* among different aspects of quality of service (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 restaurant listings, 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.

Second, user activities may involve *more than one application*, making it desirable to coordinate resource usage and adaptation policies across applications. For example, an activity that involves simultaneous video streaming and downloading email attachments may be best served when video streaming consistently uses 80% of the bandwidth and email does not attempt to go beyond 20%.

This paper presents a framework for engineering resource-adaptive systems that: (a) empower users to control tradeoffs among a rich set of aspects of QoS, and (b) coordinate resource usage among several applications. To develop such a framework, important questions need to be answered: how to represent QoS tradeoffs in a way that can be used to guide adaptation policies? How to elicit the tradeoffs preferred by users? How to allocate resources among applications, and how to coordinate their policies? What APIs must applications expose to be amenable to such coordination?

In the remainder of this paper, section 2 discusses the approach to develop such a framework, and specifically, it describes existing work on which we build. Sections 3, 4, and 5 elaborate respectively on the representation of QoS tradeoffs, on coordinating resource usage, and on enhancing applications for adaptation within the proposed framework.

Concerning evaluation, Section 3 describes a controlled user study focusing on the usability of the mechanisms for eliciting QoS tradeoffs. Sections 4 and 5 describe systems evaluations focusing on the efficiency and effectiveness of the adaptation mechanisms, respectively.

The results of the usability evaluation indicate that end-users with diverse backgrounds can understand and use the proposed model to control the adaptive behavior of applications. The results of the systems evaluations show that the implemented mechanisms are efficient, in terms of computing overhead, and effective, in the sense of making optimal decisions.

Section 6 discusses related work, and Section 7 summarizes the main points of this paper.

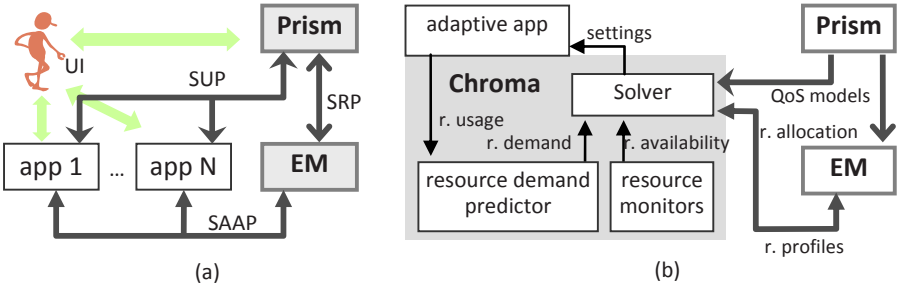


Fig. 1. The infrastructure: (a) overview of Aura, and (b) integration of Chroma

2 Architectural Baseline

Adaptability to resource limitations and variation can be promoted using two alternative architectural strategies. Either individual applications are responsible for capturing and enacting adaptation policies, or the features required to do so are factored out into a common infrastructure.

The latter approach has significant advantages both in terms of *reuse*, and of *coordinating* policies across several applications. With respect to reuse, in addition to avoiding *code* replication across applications, there is also the reuse of the *knowledge* about user preferences. For example, once the preferred QoS tradeoffs for watching a specific video stream are elicited from the user, that knowledge resides with the infrastructure and can be passed to the streaming application running on the device that happens to be convenient to the user at each moment: a cell phone, a laptop, etc.

With respect to coordination of policies, it is hard for individual applications to be aware of which other applications are being actively used, and of their resource demands and adaptation policies. Therefore, it is frequent for applications to trample each other in their quest for resources. A common infrastructure, on the other hand, may leverage a global perspective to coordinate resource allocation and policies.

Therefore, we decided to define a software infrastructure that: (a) captures models of QoS tradeoffs, (b) coordinates the resource usage across the applications supporting the user's activity, if more than one is involved, and (c) enables those applications to dynamically adjust their adaptation policies based on the models of QoS tradeoffs.

Rather than building such an infrastructure from scratch, we extended an existing infrastructure developed at Carnegie Mellon's Project Aura [4]. The remainder of this section summarizes the Aura infrastructure for ubiquitous computing [5], as well as an existing library for resource adaptation, Chroma [6], also related to Project Aura.

Aura supports a high-level notion of user activities, such as preparing presentations, or writing film reviews. Such activities may 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.

Fig. 1a shows a component and connector view of the Aura infrastructure [5]. The component called Prism captures and maintains models of user activities. Specifically, each model enumerates the services required to support the activity, how those

services are interconnected, if at all, preferences with the respect to the kinds of applications to provide each service (e.g., Emacs as opposed to vi for editing text,) and service-specific settings.

The Environment Manager (EM) component keeps track of the availability of services within an *environment*. An environment in Aura refers to the set of devices, software components and other resources accessible to a user at a particular location.

Whenever a user indicates that he or she wishes to start or resume an activity, Prism communicates the corresponding activity model to the EM using the service request protocol (SRP), and the two components negotiate the configuration that best supports the user's needs and preferences. Once an agreement is reached, the EM communicates with the applications using the service announcement and activation protocol (SAAP) to activate the services and make the required interconnections, if any. After that it passes a model of the concrete configuration up to Prism (SRP). Prism uses this model to communicate with the applications using the service use protocol (SUP) and recover the preferred settings for the activity; for example, the point at which the user was previously watching a video.

The Aura connectors (SAAP, SRP, and SUP), support the asynchronous exchange of XML messages over TCP/IP. These are peer-to-peer protocols, where each component may initiate communication, as needed.

Also related to project Aura, the Chroma library enables conventional applications to be enhanced for adaptation, provided applications can carry out their operations using different *tactics* [6]. For example, a speech recognizer may have sophisticated algorithms that deliver better results at the expense of high resource consumption, or simpler algorithms that demand fewer resources. Additionally, Chroma supports the partitioning of applications, shipping and running heavy computations in remote servers when the available resources, such as bandwidth, favor that option [7].

The internal structure of Chroma is outlined in the shaded area of Fig. 1b. It includes (a) monitors of available resources, (b) resource demand predictors based on application profiling, and (c) a solver for deciding which tactic to use at each moment. The thin arrows in Fig. 1b represent information flow among these components, as a result of method calls.

Currently supported resource monitors include history-based monitors of available bandwidth, battery charge, CPU and memory, both on the local device and on remote servers [8]. The resource demand predictor forecasts the resource demand of each tactic based on historical averages of actual demand.

Key to the workings of Chroma is a *utility function* encoded within the *solver* component. This utility function captures a specific resource-adaptation policy and is normally determined at design-time for each application. The solver determines the tactic with the highest utility, given the available resources, by exhaustive evaluation of all the tactics defined for the application. The solver is invoked by the application before carrying out each unit of work; for example, before recognizing each utterance, in the case of speech recognition, or before rendering each frame, in the case of virtual reality applications.

The research in this paper involved extending the Prism and EM components in Aura, as well as integrating Chroma with the Aura protocols and with the QoS models described in Section 3. The Aura protocols were also extended to include (a) the flow of QoS models from Prism to the EM, over the SRP, and to Chroma, over the SUP;

and (b) the flow of resource profiles from Chroma to the EM, and of resource allocation in the reverse direction, both over the SAAP. These flows are represented as the thicker arrows in Fig. 1b, corresponding to the protocols in Fig. 1a.

3 Quality–of–Service Tradeoffs

Any adaptation or optimization process is guided by a goal. In the case of adapting to resources in small mobile devices, the goal is to optimize the QoS perceived by the user. Work in this area frequently addresses conserving resources, such as battery charge, but that is just one way to optimize for service *duration*, an aspect of QoS.

The conceptual framework that we adopt takes into account that:

(1) Users may care about tradeoffs between different aspects of QoS; e.g., latency vs. accuracy.
(2) Different services may be characterized by different aspects of QoS. 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 rate and image quality.
(3) User preferences for the same service may depend on the user's activity. For example, a user may prefer high frame rate over image quality for watching a sports event over a network connection with limited bandwidth, but might prefer the opposite for watching a show on sculpture.

A simple approach to modeling user preferences is to indicate which aspect of QoS a user values the most. For example, for automatic translation, the user might indicate that response time is preferred over accuracy, and 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 end-users is a hard problem, and even more so if more than two aspects of QoS are involved.

The model we propose sits between these two extremes. User preferences are expressed as *independent* utility functions for each aspect, or dimension, of QoS. Such functions map the possible quality levels in the dimension to a normalized utility space $U \equiv [0,1]$, where the user is happy with utility values close to 1, and unhappy with utility values close to zero.

For each continuous QoS dimension the user indicates two values: the thresholds of satiation and of starvation. For example, the user might be happy with response times anywhere under 3 second, but may not accept response times over 20 seconds. This is

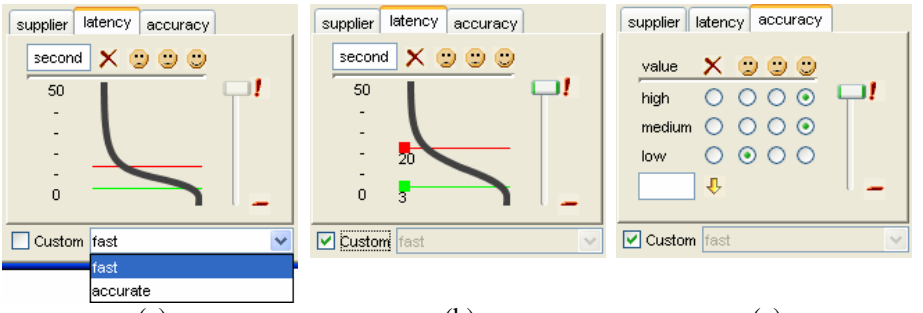


Fig. 2. QoS preferences for a language *translation* service

illustrated in Fig. 2b, where the thresholds of satiation and starvation are represented by the green (lighter) and red (darker) lines, respectively. Currently, we use sigmoid functions to smoothly interpolate between these two zones, the thresholds marking the knees of the sigmoid. The utility 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.

Preferences for discrete QoS dimensions are represented using a discrete mapping to the utility space. Fig. 2c shows an example where a table indicates the utility of each level of accuracy.

The functions for each aspect of QoS are then combined by multiplication, which corresponds to an *and* semantics: a user is 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 combines the QoS dimensions for all the services.

The relative importance of each aspect, modeled as a weight $w \in [0,1]$, is factored into the combined utility. For example, for two aspects a and b, the combined utility function is $u_a^{w_a} . u_b^{w_b}$. These weights take the value 1 by default, but may be altered using the slider bars on the right side in Fig. 2b-c.

To make it easier to use this model, we include the notion of preference *templates*. This decision is based on the principle of offering incremental benefit for incremental effort, also known as *gentle slope* systems [9]. Fig. 2a shows an example with two templates, *fast* and *accurate*. If a template is selected, the associated preferences are shown. In case a user wishes to fine-tune these preferences, he may do so after selecting the *custom* checkbox (Fig. 2b-c).

For the work herein, Prism was extended with capabilities for capturing and disseminating QoS models as illustrated above. These models are represented internally and disseminated to other components using XML, and specifically the format illustrated in Fig. 5a. The use of XML as opposed to language-specific data structures makes the models easier to exchange between components written in different languages. Prism creates user interfaces like the one in Fig. 2 dynamically, based on the XML representation of a model.

3.1 Evaluation of Usability

For the evaluation of usability, three criteria were considered: the expressiveness of the QoS models, the ease of eliciting them, and the ease of using them to control adaptation. With respect to expressiveness, our experience with multiple examples, some illustrated in the user study described below, indicate that the proposed models are expressive enough for a wide range of practical situations.

A user study investigated whether end-users can express their preferences and control adaptation using the proposed QoS models.

This study consists of using a natural language translator running on a mobile device. The quality of translation observed by users varies, since the translator runs either simple algorithms locally, or more sophisticated ones on a remote server, depending on the availability of bandwidth and of capacity in the server. To prevent limitations in the capabilities of the actual translation application (limited dictionaries, etc.) from affecting the results of the study, we replaced a human for the remote translation server. This technique is well accepted and known as a *Wizard of Oz* experiment.

The study focused on answering the following questions: first, can users understand and use 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?

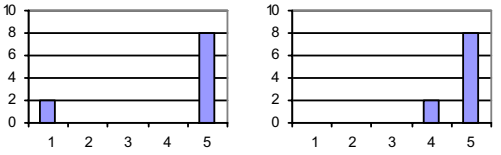
The participants were drawn from a population with homogeneous education level and age group, but diverse technical background. Ten students in the age group 18-29 were drawn among the respondents to a posting, 5 of which from computing-related fields (computer science, electrical and computer engineering, 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 30 minute introduction to the experiment, methodology and tools. Participants were asked to follow the *think aloud* protocol [10], and their voice and actions on the screen were recorded using video capturing software [11]. 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



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

Fig. 3. Likert scale evaluation of pref. specification (5-fully favorable, to 1-unfavorable)

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 – see Fig. 3a.

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 – see Fig. 3b. One of these participants suggested that an X-Y representation the tradeoff might be clearer. However, there are two reasons why that may not be such a good idea. First, it would be hard to show and manipulate tradeoffs with more than two aspects of QoS. Second, the actual tradeoff changes with the availability of resources: with plentiful resources, high levels may be attainable along all aspects; but with low resources, to privilege one aspect may have a severe impact on others.

The participants were able to interpret the effects of 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.

Fig. 4 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). When asked about how easy it was to use the interfaces in Fig. 2 to customize preferences, 5 participants were fully favorable while the other 5 thought the interfaces could be improved.

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/negative coefficient means that an increase in the x-axis corresponds to an increase/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.

Fig. 4. Regression performed on experiment data

This user study demonstrates that end-users can both define their preferences, and interpret the results of such definitions in the system's adaptive behavior. A control loop is therefore formed, enabling users to pursue concrete QoS goals. The practicality of the control loop is confirmed by the fact that all participants were easily able to achieve concrete QoS goals.

4 Coordinating Resource Usage

For this research, Aura's EM was extended with capabilities for determining and disseminating the optimal resource allocations among the applications supporting the user's activity. For that, the EM takes three kinds of inputs: models of QoS tradeoffs via the SRP, resource profiles via the SAAP, and resource estimates.

Fig. 5a illustrates the models of QoS tradeoffs corresponding to the user preferences in Fig. 2b-c. Fig. 5b illustrates resource profiles, which relate the quality levels that each application can operate at with the corresponding resource demands. Similarly to adaptive applications (see section 5,) the EM also uses resource forecasting components. However, in contrast to the fine-grained forecasts for adaptive applications, these estimates take into account large numbers of historical samples in order to forecast availability for several seconds into the future.

Given the three inputs above, the EM computes a resource allocation for each selected application, which is optimal in the sense that it maximizes the overall utility for the user's activity. Fig. 5c shows an example of resource constraints that the EM might send to one application via the SAAP.

The EM runs the algorithm for optimal resource allocation in response to changes made by the user to the QoS models, and periodically, to address trends in resource availability.

(a)

```
<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>
```

(b)

```
<service type="speechRecognition”>
  <QoSprofile>
    <QoSdim name="latency" type="float"/>
    <QoSdim name="accuracy" type="enum"/>
    <head>latency accuracy cpu bdwdth</head>
    <units>second none      %   Kbps</units>
    <point> 0.05  low      30    250</point>
    <point> 0.05  high     80    250</point>
    <point> 0.1   low      20    200</point>
    <point> 0.1   high     75    200</point>
  </QoSprofile>
</service>
```

(c)

```
<constraints>
  <rsrc id="cpu" avg="30" var="10" u="%"/>
  <rsrc id="bdwdth" avg="800" var="100" u="Kbps"/>
</constraints>
```

Fig. 5. (a) Representation of the preferences in Fig. 2; (b) example QoS profile; (c) example resource allocation

4.1 Evaluation

EM's evaluation focused on efficiency, making sure that running the resource allocation algorithm does not introduce perceptible delays for the user, and does not draw significantly from the resources available to the applications. 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 to provide those services, and when the search space of combined QoS levels reaches up to 15,000 points.

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. The optimality of decisions was verified analytically, and more details about the EM's evaluation can be found in [12].

5 Adaptive Applications

The key requirements for adaptive applications within the proposed infrastructure are: first, to comply with the resource allocation determined by the EM; and second, to enforce the QoS tradeoffs communicated by Prism, in the face of resource variations.

To reduce the costs of addressing these requirements in every application, we decided to integrate Chroma, a software layer for resource adaptation [6]. Architecturally, adaptive applications are built on top of Chroma, and there is one run-time instance of the Chroma library deployed with each application.

Integrating such applications involved wrapping them to mediate between the protocols supported by the Aura connectors and the Chroma APIs. The thicker arrows in Fig. 1b represent information flow between Prism, the EM, and Chroma, over the Aura connectors. Since Chroma expects a generic utility function to be encoded within the solver, plugging in a function that interprets the QoS models passed via the SUP (Fig. 5a) was fairly straightforward.

In the interest of space, how to enhance applications for adaptation with Chroma is not further discussed here, but details can be obtained in [6].

5.1 Evaluation

The systems evaluation presented here follows closely the experiment described in Section 3.1. 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 [13].

The utility functions provided to Chroma correspond to the *fast* and *accurate* templates introduced in Section 3. The fast template accepts medium accuracy within 1s, and the *accurate* template is willing to wait 5s for highly accurate translations. The latency thresholds for the system testing are much smaller than the ones used in the user study in Section 3.1, in which the translation was performed by a team member. 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

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

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. A more thorough validation of Chroma's ability to perform adaptation in the presence of limited resources is presented in [7].

6 Related Work

Similarly to the proposed framework, others have leveraged techniques from micro-economics to elicit utility with respect to multiple attributes. In the Security Attribute Evaluation Method (SAEM), the aggregate threat index and the losses from successful attacks are computed using utility functions [14]. The Cost Benefit Analysis Method (CBAM) uses a multidimensional utility function with respect to QoS for evaluating software architecture alternatives [15]. Our work is different from SAEM and CBAM in that it is geared towards mobile computing.

A body of work addressed battery duration in mobile devices. For example [1], presented OS extensions that coordinate CPU operation, OS scheduling, and media rendering, to optimize device performance, given user preferences concerning battery duration. The QoS models in our framework are significantly more expressive, since they support a rich vocabulary of service-specific aspects of QoS.

User studies done in mid-to-late 1990s have demonstrated that stability (e.g., absence of jitter) is more important than improvement for certain aspects of QoS [16]. Our framework recognizes the importance of these results and ensures, by explicit resource allocation, that adequate resources are available for applications to provide service while maximizing the overall utility.

Dynamic resolution of resource allocation policy conflicts involving multiple mobile users is addressed in [17] 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 on a small mobile device.

From an analytical point of view, closest to our resource allocation algorithm are Q-RAM [18], Knapsack algorithms, and winner determination in combinatorial auctions. By

integrating with generic service discovery mechanisms in the EM, our work provides an integrated framework for service discovery, resource allocation and adaptation [5].

7 Conclusions

Resource adaptation can play an important role in improving user satisfaction with respect to running sophisticated software on small mobile devices.

However, today, many applications implement limited solutions for resource adaptation, or none at all. The primary reasons for that are: (a) the cost of creating ad-hoc adaptation solutions from scratch for each application; and (b) the difficulty of coordinating resource usage among the applications. Because it is hard for an individual application to even know which other applications are actively involved in supporting a user's activity, individual applications frequently trample each other in their quest for resources.

This paper proposes a framework for resource adaptation where a number of features are factored out of applications into a common infrastructure.

First, user preferences with respect to overall QoS tradeoffs are elicited by an infrastructural component such as Prism. These models are expressed using a rich vocabulary of service-specific QoS aspects.

Second, resource allocation among applications is coordinated by another infrastructural component such as the EM. This component receives QoS profiles from applications, and efficiently computes the resource allocations that optimally support the QoS goals, given forecasts of available resources for the next few seconds.

Third, adaptation to resource variations at a time granularity of milliseconds is facilitated by a common library, such as Chroma. This library saves application 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 tactic best supports the QoS goals, given resource forecasts for the next few milliseconds.

Additionally, this paper clarifies concrete APIs that adaptive applications need to support for being integrated into the framework. These APIs are realized as XML messages, which may be exchanged within the mobile device, or across the network, if some of the infrastructural components are deployed remotely.

In summary, the proposed framework makes it easier to develop and integrate applications into coordinated, resource-adaptive systems. Furthermore, our research indicates that end-users with diverse backgrounds are able to control the behavior of such systems to achieve activity-specific QoS goals.

References

1. Yuan, W., Nahrstedt, K., Adve, S., Jones, D., Kravets, R.: GRACE-1: Cross-Layer Adaptation for Multimedia Quality and Battery Energy. IEEE Transactions on Mobile Computing 5, 799–815 (2006)
2. De Lara, E., Wallach, D., Zwaenepoel, W.: Puppeteer: Component-based Adaptation for Mobile Computing. In: USENIX Symposium on Internet Technologies and Systems (USITS), pp. 159–170. USENIX Association, San Francisco (2001)

3. Flinn, J., Satyanarayanan, M.: Energy-aware Adaptation for Mobile Applications. ACM SIGOPS Operating Systems Review 33, 48–63 (1999)
4. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P.: Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing 1, 22–31 (2002)
5. Sousa, J.P.: Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments. Carnegie Mellon University, Pittsburgh (2005)
6. Balan, R.K., Gergle, D., Satyanarayanan, M., Herbsleb, J.: Simplifying Cyber Foraging for Mobile Devices. Carnegie Mellon University, Pittsburgh (2005)
7. Balan, R.K., Satyanarayanan, M., Park, S., Okoshi, T.: Tactics-Based Remote Execution for Mobile Computing. In: USENIX Intl. Conference on Mobile Systems, Applications, and Services (MobiSys), pp. 273–286. ACM, San Francisco (2003)
8. Narayanan, D., Flinn, J., Satyanarayanan, M.: Using History to Improve Mobile Application Adaptation. In: 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), Monterey, CA (2000)
9. Myers, B., Smith, D., Horn, B.: Report of the End-User Programming Working Group. In: Myers, B. (ed.) Languages for Developing User Interfaces, pp. 343–366. Jones and Barlett, Boston (1992)
10. Steinberg, E. (ed.): Plain language: Principles and Practice. Wayne State University Press, Detroit, MI (1991)
11. TechSmith: Camtasia Studio (accessed 2008), `http://www.techsmith.com/`
12. Poladian, V., Sousa, J.P., Garlan, D., Shaw, M.: Dynamic Configuration of Resource-Aware Services. In: 26th International Conference on Software Engineering, pp. 604–613. IEEE Computer Society, Edinburgh (2004)
13. Frederking, R., Brown, R.: The Pangloss-Lite Machine Translation System. In: Expanding MT Horizons: Procs 2nd Conf Association for Machine Translation in the Americas, Montreal, Canada, pp. 268–272 (1996)
14. Butler, S.: Security Attribute Evaluation Method. A Cost-Benefit Approach. In: Intl. Conf. in Software Engineering (ICSE), pp. 232–240. ACM, Orlando (2002)
15. Moore, M., Kazman, R., Klein, M., Asundi, J.: Quantifying the Value of Architecture Design Decisions: Lessons from the Field. In: Intl. Conf. on Software Engineering (ICSE), pp. 557–562. IEEE Computer Society, Portland (2003)
16. Wijesekera, D., Varadarajan, S., Parikh, S., Srivastava, J., Nerode, A.: Performance evaluation of media losses in the Continuous MediaToolkit. In: Intl. Workshop on Multimedia Software Engineering (MSE), Kyoto, Japan, pp. 60–67. IEEE, Los Alamitos (1998)
17. Capra, L., Emmerich, W., Mascolo, C.: CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. IEEE Transactions on Software Engineering 29, 929–945 (2003)
18. Lee, C., Lehoczky, J., Siewiorek, D., Rajkumar, R., Hansen, J.: A Scalable Solution to the Multi-Resource QoS Problem. In: IEEE Real-Time Systems Symposium (RTSS), pp. 315–326. IEEE Computer Society, Los Alamitos (1999)