# Tactics-Based Remote Execution

Rajesh Krishna Balan
Carnegie Mellon University
rajesh@cs.cmu.edu

## 1 Introduction

*Remote execution* can transform the puniest mobile device into a computing giant. This would enable resource-intensive applications such as natural language translation, speech recognition, face recognition, and augmented reality to be run on tiny handheld, wearable or body-implanted platforms. Nearby compute servers, connected through a low-latency wireless LAN, can provide the CPU cycles, memory, and energy needed for such applications.

Unfortunately, two annoying facts cloud this rosy future. First, the partitioning of an interactive application into local and remote components that achieves good application performance is highly application-specific and platform-specific. Since mobile hardware evolves rapidly, this partitioning changes on the timescale of months rather than years. Incorrect partitioning can result in sluggish and intolerable interactive response. Hence, a tight and ongoing coupling between application developers and hardware platform developers appears inevitable. Second, matters are made worse by the fact that mobile environments exhibit highly variable resource availability. Bandwidth, energy and availability of compute servers can change on the timescale of minutes or hours, as a user moves to different locations. Re-partitioning an application for changed operating conditions at this timescale is therefore essential. These considerations suggest that an automated approach to partitioning applications for remote execution is necessary. However, partitioning an application automatically may result in bad performance if the intrinsic nature of the application is not factored into the partitioning decision.

Another factor that clouds this future is that developing mobile applications is especially difficult because they have to be adaptive [14, 21, 38]. The resource constraints of mobile devices, the uncertainty of wireless communication quality, the concern for battery life, and the lowered trust typical of mobile environments all combine to complicate the design of mobile applications. Only through dynamic adaptation in response to varying runtime conditions can applications provide a satisfactory user experience. Unfortunately, the complexity of writing and debugging adaptive code adds to the application software development time.

The key questions that this thesis is addressing can be expressed as follows:

1. Can automated dynamic re-partitioning (to respond effectively to changes in mobile environments) be reconciled with the need to exploit application-specific knowledge (to achieve good application performance)?
2. Is it possible for application developers to easily develop adaptive applications? For this question, I will be concentrating on the problem of modifying existing applications to make them adaptive.

The key insight that allows both these problems to be solved is that *the knowledge about an application relevant to remote execution can be captured in compact declarative form that is very small relative to code size.* More specifically, the full range of meaningful partitions of an application can be described in a compact external description called *remote execution tactics* or just *tactics* for brevity. Thus, the tactics for an application constitute the limited and controlled exposure of application-specific knowledge necessary for making effective partitioning and placement decisions for that application in a mobile computing environment.

## 2 Thesis Statement

**The full range of meaningful partitions of an application can be described in a compact external description called *remote execution tactics* or just *tactics* for brevity. This enables the development of powerful remote execution systems that can automatically at runtime, choose remote execution partitions, from the list of tactics, that provide good application performance. Finally, the use of tactics is consistent with sound software engineering principles that can decrease the time needed to develop mobile applications for new hardware platforms.**

## 2.1 Why are Tactics Useful?

There have been a number of remote execution systems such as Abacus [2], Coign [19] and Condor [5]. They perform well in environments where resource availability does not change between the time the system decides how to remotely execute an application and when it performs the remote execution. However, this assumption comes under fire in mobile environments. These environments are characterized by highly variable resource conditions that change on the order of seconds [10, 14, 37].

To achieve the good application performance, a remote execution system should pick good strategies for remote execution for any particular resource condition. But how do we determine what that good strategy is? Previous solutions to tackle this problem have fallen between two extremes. At one extreme, we have a method known as static partitioning. In this method, the application is manually modified to use remote execution and the precise method of remote execution along with the exact servers to be used is hard-coded into the application. This method is conceptually easy to understand and relatively easy to implement. However, this static method is extremely ineffective in mobile environments where resources change dynamically. At the other extreme, we have full dynamic partitioning. In this method, a runtime system is built that analyzes the current environment and determines the current resource availability. The runtime then tests every possible way of remotely executing the application and picks the solution that achieves best performance. This method is highly effective but extremely difficult to implement. Building such a runtime is hard and searching all possible remote execution possibilities is frequently intractable in practice. The perfect solution is to build a system achieves the performance of full dynamic partitioning with the ease of static partitioning. Unfortunatelt, a general solution that achieve this seems to be impossible due to the infinite ways in which applications can be partitioned. However, I claim that it is possible to achieve this solution for handheld devices because of the following claim:

*For a large number of useful mobile applications, the number of useful ways of splitting the applications for remote execution is small.*

These useful ways of splitting the application are called the *tactics* of the application. Tactics are specified by the application developer and are high level descriptions of meaningful module-level partitions of an application. Given this information, it is possible to build a remote execution system that adapts application performance by picking the appropriate partitions according to the current resource availability in the environment. In my thesis, I will validate the claim by showing that it is possible to specify the tactics of a large number of useful mobile applications.

Tactics also allow the remote execution system to use extra resources in the environment to automatically improve application performance. This use of tactics is explained further in Section 3.4.

## 2.2 Need for Adaptive Remote Execution Systems

This work is one of the first to look at developing a remote execution system that can automatically adapt application behaviour depending on the resources available in a mobile environment. It builds upon the work of the Odyssey [29] and Spectra [13] projects.

However, why is such a system useful? Given the exponential increase in computational power (as predicted by Moore's Law), why can't mobile devices just execute all their applications locally and at full quality? Why do we need systems that execute applications on remote servers and / or change their *fidelity* [29] [1] in order to achieve acceptable performance? After all, doesn't Moore's Law mean that adaptive runtime systems will become obsolete in 5 years time when mobile devices become much more powerful?

The answer is no and there are three reasons for this. First, the mobile device market is driven by the need to make devices that are smaller and lighter than the previous generation in order to satisfy the demands of the consumer. Satisfying these requirements usually requires sacrificing resources (like disk space) in order to meet the space and weight criteria. Second, battery performance has not followed Moore's Law and has remained fairly constant over the years. As such, when the devices become smaller, the device manufacturers will have to use slower less powerful but more power-efficient CPUs and smaller memories in order to achieve decent battery lifetimes. Third, wireless bandwidth has not been increasing at the same rate as wired bandwidth. For example, wireless bandwidth has increased from 2Mb/s to 11Mb/s with 54Mb/s forthcoming. However, wired bandwidth to the desktop has increased from

[1] Fidelity refers to an application specific metric of quality. For example, speech recognition has higher fidelity when using a large vocabulary rather than a small vocabulary. Fidelity ranges from 0 to 1, with 1 being the best quality and 0 the worst.

10Mb/s to 100Mb/s and now 1000Mb/s in the same period of time.

However, the desktop market is not constrained by battery lifetimes, by space and weight constraints or by bandwidth constraints. As such, desktop machines will still be following Moore's Law and the gap between the computational capabilities of the desktop machine and the mobile device will remain large even in the foreseeable future.

These gaps in resource availability between desktop machines and mobile devices is one of the reasons why adaptive runtime systems are necessary. Many applications are developed primarily for the desktop market. As such, they require the computational and bandwidth capabilities of the desktop machine. Mobile users wanting to use such applications on their mobile devices will require a remote execution system that can run these applications on remote servers.

It could be said that remote execution systems are unnecessary because application writers are developing applications specifically for mobile devices. However, these applications tend to be scaled down versions of desktop applications. They usually are not as powerful as the desktop versions. A remote execution system would allow the use of these more powerful desktop applications on mobile devices that would otherwise not be able to adequately run these applications.

Finally, there are a large class of computationally intensive interactive applications that are important for mobile users. These include language translation, speech recognition and augmented reality applications. However, these applications require computational power that usually exceeds a wearable mobile device's capabilities. Remotely executing these applications will allow them to be used on these wearable mobile devices.

All these factors strongly indicate that creating a powerful adaptive runtime system for mobile devices is a valuable endeavour. However, the challenge with such systems is to build them such that they are able to quickly react to resource changes in the highly variable mobile environment. I claim that tactics allows me to build such a system. Furthermore, a key difficulty with using existing remote execution systems is that modifying applications to use them tends to be prohibitively expensive. The second part of my claim is that tactics allows me to use software engineering techniques to reduce the time needed to develop adaptive mobile applications.

## 2.3 Reducing Adaptive Mobile Application Development Time

The proliferation of task-specific mobile and wearable devices with short lifetimes places severe stress on the development and maintenance of adaptive mobile applications. A critical factor limiting the commercial success of such a device is the software development time needed to create useful applications for it. The longer this development time, the shorter the useful life of the device in the marketplace. Slow software development can make the device obsolete by the time it emerges as a product. Business opportunities are measured in months rather than years in this fast-paced field.

Developing mobile computing applications is especially difficult because they have to be adaptive [14, 16, 21, 38]. The resource constraints of mobile devices, the uncertainty of wireless communication quality, the concern for battery life, and the lowered trust typical of mobile environments all combine to complicate the design of mobile applications. Only through dynamic adaptation in response to varying runtime conditions can applications provide a satisfactory user experience. Unfortunately, the complexity of writing and debugging adaptive code adds to the application software development time.

*How can we reduce the software development costs of adaptive mobile applications?* This is a fundamental question that I address in this thesis. My proposed solution is based on three observations that are derived from first-hand experience with building adaptive mobile applications.

- First, most applications for mobile devices can be created by modifying existing applications rather than writing new applications from scratch.
- Second, the modifications for adaptation typically affect only a small fraction of total application code size. Much of the complexity of implementing adaptation lies in understanding the base code well enough to be confident of the changes to make.
- Third, the changes for adaptation can be factored out cleanly and expressed in a platform-neutral manner.

My approach can be summarized as follows:

- A lightweight semi-automatic process for customizing the API used by the application to interface with the adaptive runtime system Such customization is targeted to the specific adaptation needs of each application.
- Next, a tool for automatic generation of code stubs that maps the customized API to the specific adaptation features of the underlying mobile computing platform will be provided.
- Finally, the run-time support for monitoring resource levels and triggering adaptation will be factored out of applications and into a set of operating system extensions for resource adaptation.

Each of these components plays an important role in the overall effectiveness of the approach. The first component (semi-automatic process) amortizes the effort of understanding an application and extending it for adaptation. The second component (stub generator) insulates application code from frequent changes of the underlying mobile computing platform. The third component (OS support) allows a clean separation of policy and mechanism — the OS monitors resource levels and triggers adaptation, but it is the individual applications that decide how to adapt. OS support also helps ensure that the adaptations of multiple concurrently executing applications do not interfere with each other.

This approach complements traditional software engineering techniques such as code modularity and separation of concerns [33]. In addition, the approach takes into account the context-sensitive nature of adaptation policies. In other words, high level attributes such as a user's location, physiological state, and cognitive load are often important factors in determining how a low-level adaptation decision should be made [41]. This implies a bridging of system layers that is not common in non-mobile applications.

In this thesis, I am concentrating solely on reducing the cost of developing adaptive applications. I am not tackling other software engineering issues even though, I claim that, my methods also facilitate other nice software engineering properties (like making software maintainence easier) due to the use of modularity and separation of concerns.

## 2.4 Validation

I plan to validate this thesis in the following steps:

- Define the semantics and syntax of tactics. This step involves clearly defining what the tactics of an application are and what information they convey. A language for specifying tactics also needs to be developed. This language has to be simple enough for application writers to easily use yet be powerful enough to capture the full semantics of tactics.
- Develop a prototype remote execution system that uses tactics. This prototype should demonstrate that it is possible for applications to specify their tactics to a remote execution system using the developed language. I have already started implementing this prototype system and it is called *Chroma*. In the rest of this proposal, I will be using Chroma to refer to the prototype being built to validate the effectiveness of tactics.
- Demonstrate that tactics provide enough information to partition applications effectively by showing how the semantics of tactics allows Chroma to automatically provide good performance for applications.
- Show that tactics are applicable to a wide range of applications. The verification of this step will involve defining the tactics for a large number (about 5-10) of useful mobile applications.
- Verify the effectiveness of the software engineering methods in reducing the application development time by quantifying the time needed to add the mobile applications used for validating the generality of tactics to Chroma.

There are several problems that need to be solved to reach this goal:

- Describing tactics requires some kind of language. However, what should this language look like? Will a simple language suffice?
- Tactics describe the useful partitions of an applications. However, what functionality does Chroma require in order to successfully pick the right partition to use at runtime? At the very least, Chroma will need to know the current resource availability in the system along with the expected resource usage of each partition specified in the tactics. However, how should this information be collected and saved? Also, how should Chroma combine these two pieces of information to decide which partition to use?

- Chroma uses remote servers to execute the remote partitions specified by tactics. However, how should these remote servers be discovered?
- To be useful to the user, Chroma needs to know the user's preferences. This is because the user may have reasons to prefer certain partitionings over other partitionings specified in the tactics and these preferences could change dynamically. However, obtaining these preferences from the user automatically is a known hard problem [18]. Software such as Prism [41] operate at a higher layer than Chroma and attempts to capture user preferences. Prism's goal is to achieve the task requested by the user. By exchanging information between Chroma and Prism, it may be possible for Chroma to obtain user preferences via Prism. Prism could inform Chroma about the user's preferences for each of the applications that make up that task and leave the task of adapting each application according to the current resource availability to Chroma. However, defining the interface between Chroma and Prism is a non-trivial task.
- The effectiveness of tactics in a remote execution system needs to be properly verified. To do this, a large number of applications need to be modified to use Chroma. Their tactics need to be properly specified and the improvements in application performance achievable by Chroma needs to be quantified. Unfortunately, adding existing applications to Chroma could be time consuming as it requires making each application adaptive. As such, developing software engineering methods that lower the time needed to add existing applications to Chroma is crucial to make this aspect of the verification manageable.

The remainder of this document is organized as follows. Section 3 describes the portion of this work that has been completed, or largely completed — a language for describing tactics and a 1st-pass implementation of Chroma that uses tactics. It also describes the 4-step process that has been developed to make the development of adaptive mobile applications faster. Section 4 is my plan for the remainder of the thesis work: it describes my approach towards the problems mentioned here — extending Chroma to make better use of tactics, improving the software engineering methods, and the verification of both aspects of my thesis statement. Section 5 describes five scenarios that this thesis hopes to enable. Section 6 is an itemized list of work items for the proposed thesis work. Section 7 describes related work while Section 8 outlines the expected contributions of this work to the field. Finally, Section 9 provides a time-line for the thesis work.

# 3 Completed Work

## 3.1 The semantics and description of Tactics

### 3.1.1 Assumptions

The power of tactics lies in their ability to distill the useful ways of partitioning an application for remote execution. However, first a model of the applications targeted and the kind of remote execution being performed by those applications needs to be created.

In this thesis, I consider the class of computationally-intensive interactive applications as they contain a large number of useful mobile applications. Examples include speech recognition, natural language translation and augmented reality applications. These are the kinds of applications that have been envisioned as being key mobile applications in the near future [39, 46]. I explain how I plan to select the applications I will be considering in Section 4.1.1.

For this class of applications, I claim that coarse-grained remote execution is sufficient. These applications have a user in the loop and as such can usually sustain latencies of up to 1-2 seconds. This is in contrast to systems like Java RMI [42] that perform fine-grained remote execution on the order of microseconds. I will be using remote procedure calls (RPC) [7] to perform this coarse-grained remote execution. Furthermore, I assume that each RPC is fully self contained and has no side effects. However, the use of RPCs and the lack of side effects is not intrinsic to tactics but rather simplifications for the purpose of this thesis.

Finally, each application is made up of *operations*. An operation is an application-specific notion of work such as translating a sentence for a language translator or reducing a scene for an augmented reality application. Each operation can have its own unique set of useful partitions. As such, the complete set of tactics for an application will contain tactics for each of the operations supported by that application.

### 3.1.2 Semantics and Language for Describing Tactics

Given the assumptions in Section 3.1.1 a simple language was developed for allowing application writers to specify the tactics for their applications. This language consists of two portions; The first part consists of a description of the various procedures in the application that can be remotely executed. These procedures are the set of RPCs for that application. The inputs and outputs to each RPC is specified and each RPC can be executed either remotely or locally and this decision is made at runtime. The second part of the language is used to describe the how these RPCs can be usefully combined.

The RPCs can be combined in the following ways:

1. Sequential dependencies between RPCs are allowed. I.e., it can be specified that RPC A has to be finished before RPC B.

2. It is possible to specify that a group of RPCs have no dependencies between them and can executed in parallel. For example, it can be stated that RPCs A, B and C can all be executed in parallel. However, in my initial prototype, if a group of RPCs is specified as being able to be executed in parallel, that group must be followed by a single sequential RPC. I.e., if it is stated that RPCs A, B and C can be executed in parallel, than all 3 RPCs must be followed by the same sequential RPC D. It is not possible for RPC A to be followed by RPC D while RPCs B and C are followed by some other RPCs. This limitation of the prototype will be fixed if I find real applications that require more complicated patterns of specifying RPCs.

Each of these RPC combinations is a separate tactic and fully describes one way of combining RPCs to complete an operation. An operation can have many tactics that may differ in their fidelity and resource usage. As such, at runtime, the remote execution system picks the tactic that provides the highest fidelity while satisfying all resource constraints. A formal description of the syntax of this language used for describing tactics is provided in Appendix A.

The data dependencies between RPCs can be determined by analyzing the inputs and outputs of each RPC as specified in the tactics description (since I assume no side effects). Each of the individual remote calls that make up a particular tactic can be run either locally or on any remote server. This decision is made at runtime. If necessary, the application developer can constrain a particular tactic to use particular servers for its operation (This is shown in Figure 1 for the *dict* tactic).

Even though the tactics may differ in their resource usage and fidelity, each tactic is guaranteed to produce a proper result for the given operation if the remote calls are performed in the order specified by the tactic (I assume no side effects as mentioned in Section 3.1.1). Since the data dependencies and ordering between remote calls is fully specified by the tactic description, it is possible to automatically parallelize the execution of these remote stages. This additional power of tactics is explained further in Section 3.4.

### 3.1.3 Example Descriptions

Figure 1 shows the description of the tactics for an example application. This application is a natural language translator called Pangloss-Lite [15]. Pangloss-Lite has three different translation engines; Glossary (gloss), Dictionary (dict) and Example-based (ebmt). Each of these translation engines can be executed independently of each other. Using more engines results in higher quality results. The output of each engine is fed into a language modeller (lm) that combines the various outputs and creates the final translation.

The three translation engines and the language modeller are specified as four RPCs (*server_gloss*, *server_dict*, *server_ebmt* and *server_lm*) that can be remotely executed. The four RPCs can be combined in seven useful ways as shown by the seven tactics (*gloss*, *dict*, *ebmt*, *gloss_dict*, *gloss_ebmt*, *dict_ebmt* and , *gloss_dict_ebmt*). The & denotes a sequential dependency between RPCs while RPCs inside brackets can be executed in parallel. For Pangloss-Lite, we see that the seven RPCs are created as follows; there are seven different ways of using one or more of the three translation engines. Each of these engines is independent of each other and hence can be executed in parallel with other engines. The output of all the engines must be sent to the language modeller for final processing. The tactics *dict*, has also specified that the *server_dict* RPC should be executed on a specific remote machine (gs129.sp).

In Figure 2, we see the description of the tactics for Janus [44], a speech recognition program. Janus performs speech-to-text translation of spoken phrases. Recognition can be performed at either full or reduced fidelity. The reduced fidelity uses a smaller, more task-specific vocabulary that limits the number of phrases that can be successfully recognized but requires less time to recognize a phrase.

```
RPC server_gloss (IN string line, OUT string gloss_out);
RPC server_dict  (IN string line, OUT string dict_out);
RPC server_ebmt  (IN string line, OUT string ebmt_out);
RPC server_lm    (IN string line, IN string ebmt_out,
                  IN string dict_out, IN string gloss_out,
                  OUT string translation);

DEFINE_TACTIC gloss   =      server_gloss & server_lm;
DEFINE_TACTIC dict    =      server_dict@gs129.sp & server_lm;
DEFINE_TACTIC ebmt    =      server_ebmt & server_lm;
DEFINE_TACTIC gloss_dict =   (server_gloss, server_dict) & server_lm;
DEFINE_TACTIC gloss_ebmt =   (server_gloss, server_ebmt) & server_lm;
DEFINE_TACTIC dict_ebmt  =   (server_dict, server_ebmt) & server_lm;
DEFINE_TACTIC gloss_dict_ebmt = (server_gloss, server_dict, server_ebmt) & server_lm;
```

Pangloss-Lite has four RPCs and seven tactics (ways of combining the four RPCs) that are listed after the DEFINE_TACTIC keyword. These seven tactics give different ways of combining the remote calls (listed after the keyword RPC) for this application. Each of these calls can be executed locally or at a remote server and this is determined at runtime by Chroma

Figure 1: Tactics for Pangloss-Lite

```
RPC do_full_recognition
               (IN string utterance,
               OUT string translation);

RPC do_reduced_recognition
               (IN string utterance,
               OUT string translation);

DEFINE_TACTIC full_recognition =
                do_full_recognition;

DEFINE_TACTIC reduced_recognition =
                do_reduced_recognition;
```

The tactics declaration for Janus contains two remote calls (do_full_recognition and do_reduced_recognition) that can be run either locally or remotely.

Figure 2: Tactics for Janus

Janus has two remote calls that can be executed either locally or remotely. These two possible ways of executing Janus are captured by Janus's tactics, as shown in Figure 2. The tactic `full_recognition` uses the full fidelity vocabulary to do the recognition while the tactic `reduced_recognition` uses the reduced fidelity vocabulary to do the recognition.

Finally, in Figure 3, we see the tactics for Face [40]. Face is a program that detects human faces in images. It is representative of image processing applications of value to mobile users. Face can potentially change its fidelity by degrading the quality of the input image.

Face can be run either entirely locally or entirely remotely. In both cases, it runs the exact same remote procedure and it has no other modes of operation. It thus has only one tactic and this is shown in Figure 3.

In all three application examples, we see that the description can be divided into a list of RPCs followed by the various useful methods of combining these RPCs. These three examples also provide evidence that the language is easy to use and is able to capture the semantics of tactics. However, the language can only be fully verified by defining the tactics for a larger number of applications (explained in Section 4.1).

```
RPC detect_face (IN file in_image_name,
             OUT file out_image_name);

DEFINE_TACTIC detect = detect_face;
```

Face has only one remote call (detect_face) that can be run either locally or remotely. This is captured by its single tactic.

Figure 3: Tactics for Face

## 3.2 Chroma

### 3.2.1 Overview

The remote execution system being built to validate this thesis is called *Chroma*. Chroma is based on the Odyssey [29, 12] adaptive runtime system. It incorporates the energy, CPU, bandwidth and file cache resource adaptation capabilities of Odyssey. Like Odyssey, Chroma uses application-aware adaptation (adaptation that requires modifications to the application source). The key new functionalities that are being added to Chroma are:

- the use of *tactics*
- a remote execution system that adapts according to the number of remote servers in the environment
- a stub generator that makes it easier to add applications to Chroma
- integration with a task layer so as to better deal with user preferences
- a set of tools and methods to make adding legacy applications to Chroma faster

## 3.3 Chroma Design

In this section, the design of Chroma is described. Building Chroma required two main components:

- A way of describing tactics. This has already been described in Section 3.1.2
- A method for selecting the particular tactic to use in a given situation.

### 3.3.1 Tactic Selection

At runtime, Chroma needs to decide for a particular application which tactic to use and where to execute it. For example, if Chroma picks the tactic `gloss_ebmt` (Figure 1) for Pangloss-Lite, it will also have to decide whether to execute the `server_gloss`, `server_ebmt` and `server_lm` remote calls of this tactic locally or remotely. Chroma's goal is thus to decide on a *tactic plan*. A tactic plan comprises of a tactic number (denoting which tactic to use), along with a list that specifies the server to use for each RPC in that tactic. Chroma enumerates through all possible tactic plans and picks the best one for the given resource availability.

To be able to do this, first, Chroma needs to be able to predict the resource usage of each tactic plan. Second, Chroma has to measure the current resource availability. Third, Chroma requires guidance from the user about how to tradeoff resources in order to pick the best tactic. For example, the user may specify that bandwidth usage should be minimized (possibly because of pricing issues) or that the system should conserve battery power as much as possible. Given these three things, Chroma will be able to decide on the best tactic plan for the particular operation from the user's viewpoint.

### 3.3.2 Resource Prediction

For a given operation and tactic plan, Chroma needs to be able to predict the resources the tactic plan will require. For this, I will use Narayanan's resource demand predictors [28]. The key idea here is that the resource usage of a tactic plan can be predicted from its recent resource usage. The demand prediction mechanisms are initialized by off-line logging. At runtime, these predictors are updated with online monitoring and machine learning to improve accuracy. I do not plan to extend this aspect of Chroma beyond what is already provided by Narayanan's work.

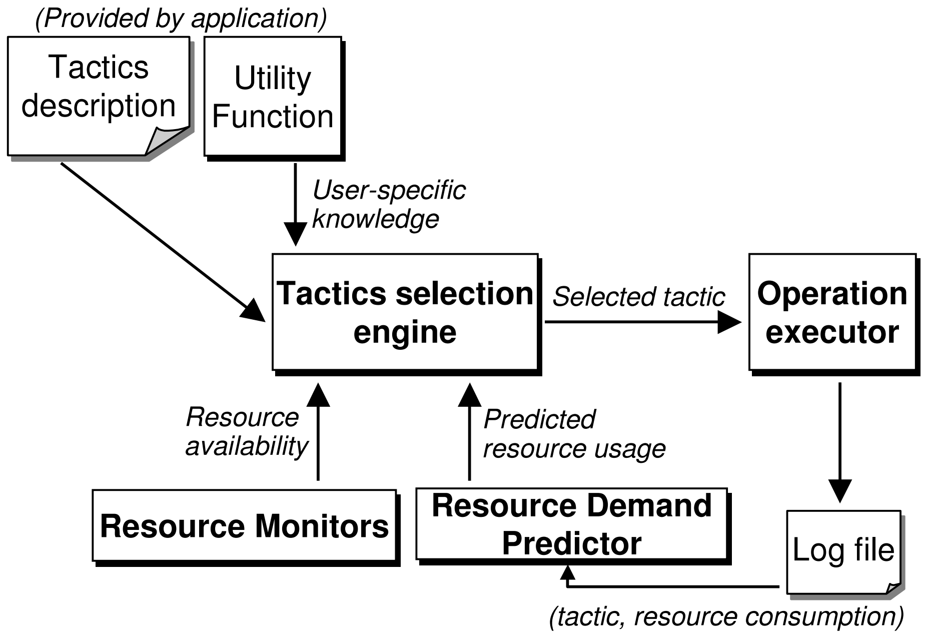


Figure 4: Choosing a Tactic

#### 3.3.3 Resource Monitoring

Chroma uses multiple resource measurers to determine current resource availability. These resource measurers currently measure memory usage, CPU availability, available bandwidth, latency of operation, file cache state and battery energy remaining. Chroma also has mechanisms to retrieve resource availability information from remote servers.

#### 3.3.4 Additional User-specific Knowledge

An operation can have many tactics, each of which has a different resource usage and fidelity. At runtime, Chroma has to decide the best tactic plan to use for a given operation and resource availability. Chroma can determine the current resource availability and the resource demand of the various tactic plans using the two components described earlier. But to effectively match the resource demand with the resource availability, Chroma needs to trade off resources for fidelity. How to perform this tradeoff is frequently context sensitive and thus dynamic. For instance, would the user of a language translator prefer accurate translations or snappy response times? Should an application running on a mobile device use power-saving modes to preserve battery charge, or should it use resources liberally in order to complete the user's task before he or she runs off to board their plane? That knowledge is very hard to obtain at the application level as it is user-specific and not application-specific.

Chroma is provided with these user-specific resource tradeoffs in the form of *utility functions*. A utility function is a user-specific function that quantifies the tradeoff between two or more attributes. These utility functions can be either provided directly by the application or by Prism. These utility functions will allow Chroma to optimize the tactic selection for other user specified metrics like conserving battery power or minimizing network bandwidth.

#### 3.3.5 Selection Process

Figure 4 shows how all the components work together. Chroma determines the predicted resource demand for each tactic of the current operation by querying the resource prediction component. At the same time, Chroma determines the available resources via the resource monitoring component. These resource monitors also query any available remote servers to determine the resource availability on those servers. This information is necessary as the latency of the tactic is determined by where each individual remote call in that tactic is being executed. Determining resource availability on demand can be a very time consuming operation. Hence, to improve performance at the cost of accuracy, the resource monitors perform these queries periodically in the background and cache the results.

Chroma iterates through every possible tactic plan and picks the best tactic plan to use for this operation. It does this by picking the tactic plan that maximizes the utility function specified by the user (As described in Section3.3.4.

The tactic plan is then executed and its resource usage is logged to refine demand prediction for the future. This brute force method works well for a small number of tactics (less than 30 for a PDA), however it will become computationally infeasible when the number of tactics increases. However, we claim that the number of useful tactics for computationally-intensive interactive applications are small enough to allow this brute force tactic selection mechanism. This claim will be empirically validated as more applications are integrated into the systems. We also looking at using other solvers that are less computationally demanding [23].

## 3.4 Over-Provisioned Environments

The discussion so far has focused on the assumption that the environments we are in are mostly resource constrained. However, environments such as smart rooms, may be *over-provisioned*. Over-provisioned environments are characterized as having more computing resources than are needed for normal operation. It is highly valuable to have a system that works well if resources are scarce but is able to immediately make use of over-provisioning if it becomes available.

Tactics provide us this ability as they allow us to automatically use extra resources to improve performance. This is possible because tactics provides the knowledge of the remote calls needed by a given operation and the data dependencies between them. Chroma can use this knowledge to execute remote calls opportunistically to improve performance in three different ways:

First, Chroma can make multiple remote execution calls (for the same operation) to remote servers and use the fastest result. For example, Chroma can execute the glossary engine of Pangloss-Lite at multiple servers and use the fastest result. Chroma knows that it can do this safely because the description of the tactics makes it clear that executing the glossary engine is a stand-alone operation and does not require any previous results or state.

Second, Chroma can perform the same operation but with different fidelities at different servers. Chroma can then return the highest fidelity result that satisfies the latency constraints of the application. For example, Chroma can execute multiple instances of the `ebmt` engine of Pangloss-Lite in parallel at separate servers (all with different fidelities) and use the highest fidelity result that has returned before a specified amount of time.

Third, Chroma can split the work necessary for an operation among multiple servers. It does this by decomposing operation data into smaller chunks and shipping each chunk to a different remote server. Chroma uses hints from the application to determine the proper method of splitting operation data into smaller chunks.

Initial experimentation has shown that these optimizations can provide substantial performance improvements for applications. In addition, these improvements can be provided by Chroma automatically when extra servers become available without needing to inform the application. However, more work needs to be done to quantify the benefits of using extra resources as well as improve Chroma's ability to automatically to use these extra resources in the best possible manner for a given application. Finally, I need to develop mechanisms that will ensure that these extra servers are used by multiple Chroma clients in a fair and distributed manner.

## 3.5 Integration with Prism

One key observation of this work is that determining appropriate adaptation policies is critically dependent on the ability to capture user expectations. Capturing user expectations is a hard problem that is being addressed in a layer called *Prism*. Prism treats user tasks as first class entities and interacts with context-aware components to assess the physical context around the user. It determines the most accurate models of user expectations using stochastic techniques to correlate the current user context to past experiences. By capturing user expectations outside of applications, we enable the reuse of user expectation models. This allows the migration of user tasks in pervasive computing environments [41]. Prism is being developed by another Ph.D. student, Joao Pedro Sousa, and I plan to interface Chroma with Prism.

However, a key research question is determining the proper interfaces between Chroma and Prism. How much information needs to be shared between Chroma and Prism in order to achieve the best possible performance from the users perspective? I aim to explore this question and come up with design guidelines that can be used to guide future efforts to couple user level adaptation with operating system / application level adaptation.

## 3.6 Reducing Development Cost

Much of the cost of building and maintaining adaptive applications comes from the low-level at which adaptation enhancements are captured. Understanding the required adaptation features and implementing them over the APIs offered by the underlying platform is a costly process. Currently, there is no effective way to preserve such investment except in the form of embedded code modifications. These are hard to maintain in the face of the fast rate of release of new platforms.

To solve this problem, I created a high-level declarative language (formally described in Appendix B) that is used to describe the adaptation aspects of an application. That description is then compiled and a code stub is generated. This code stub creates a *customized* API for the application, which is derived from the high-level description of the adaptation requirements. This customized API is much closer to the application's needs than a generic low-level adaptation API, and thus makes it much easier to integrate the adaptation aspects with the bulk of the application code.

Furthermore, applications in the same domain, say video players, are likely to have very similar adaptation requirements. Hence, adaptation descriptions can be reused among such applications. For example, it would be easier to extend the next video player for adaptation once we've completed the first one.

By having a compiler-based approach, it is possible to amortize the effort of re-targeting a set of adaptive applications to a new platform. After all, it is easier to re-target the code generation of a compiler than to modify each application manually.

The hypothesis here is that it will be easier to re-target the code generation for a new platform, and recompile all the applications, than re-targeting every application.

The specific runtime targeted by my stub generator is Chroma. However, the use of a stub generator allows me to potentially use any other OS or adaptation middleware [2, 5, 19, 29], without changes to the application source or description files. I merely have to change the stub generator to produce code targeted for these new runtime systems.

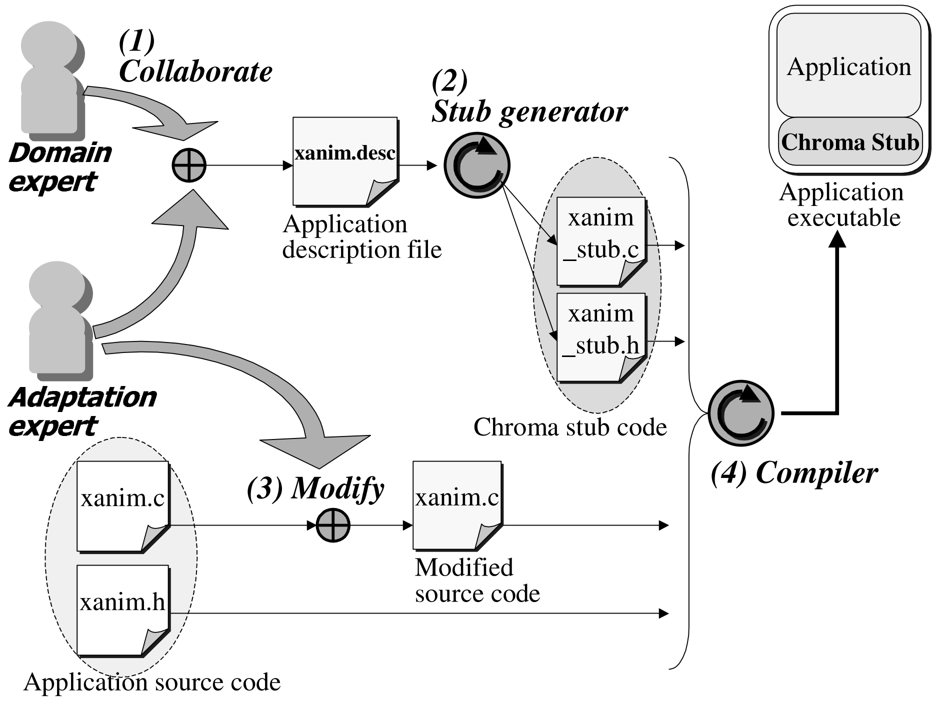


Figure 5: Process for adding adaptation to an application

### 3.6.1 4-step Process to Reduce Development Time

The 4-step process that I have developed for use by application developers to create their application descriptions and the corresponding application stubs is shown in Figure 5.

1. The adaptation expert collaborates with a domain expert to produce an *application description* that captures the information necessary for the application to be adaptive. For instance, the description for XAnim contains the adaptive variables relevant to adaptive video playing: frame rate, encoding, frame quality, height, and width. This description is platform-independent, and can be reused for other applications that provide adaptive video playing capabilities. I.e., it applies equally to XAnim and to MediaPlayer, to Linux and to Windows.

2. A *stub generator* compiles the application description into a set of stubs that interface between the application and the underlying runtime support.

3. The application is modified to invoke the functions provided by the stub layer. This step is manual, and must be done for each application. However, these changes are small and localized as demonstrated in our case studies, and this fact makes it easy to preserve the adaptation enhancements in new releases of the applications, as described in Section 1.

4. The application source code and stub are compiled, and linked together to form the application binary. When executed, this binary invokes the runtime support layer to make adaptive decisions.

## 4 Planned Work

I have successfully built the first version of Chroma, that uses tactics to perform remote execution of applications. Chroma also automatically uses extra remote servers to improve application performance. The performance achieved by this 1st-pass implementation of Chroma are detailed in my MobiSys 2003 paper [4]. However, I still need to implement the following components to fully validate the claims of my thesis statement.

### 4.1 Validation / Modification of Tactics Language through Multiple Application Case Studies

I plan to verify the language used for specifying tactics by using it to describe the tactics for a large number of applications. These applications will be chosen to represent a broad range of applications that might be useful for mobile users. Based on the experience gained in describing these applications, the tactics specification language will be extended and/or modified as necessary.

#### 4.1.1 Selection of Applications

For the validation of the generality of tactics to be sufficient, it is important to choose a large range of applications that adequately map the space of computationally-intensive interactive applications. Currently, I have used 3 research applications that were all developed at Carnegie Mellon University (CMU). Since they were all from the same institution, using them alone may not be sufficient to verify the generality of tactics. For example, all these applications were single-threaded and as such, the applicability of tactics to threaded applications is unclear. Also, it is possible that the applications developed at CMU were particularly suited to the use of tactics and are not representative of computationally-intensive interactive applications being developed elsewhere.

Hence, to effectively verify the generality of tactics, I plan to use various non-academic open-source applications like mplayer (a video player). If possible, I will also use research applications developed at other universities. The applications chosen should exhibit different programming styles like the use of threads etc. This diverse selection of applications and programming styles will help demonstrate the applicability of tactics to a large set of computationally-intensive interactive applications. I hope to end up with at least 5 to 10 different applications comprising a good mix of discrete and continuous applications from both academic and non-academic sources exhibiting a range of programming styles.

### 4.2 Support for Continuous Applications

Currently, Chroma only supports discrete applications. These are applications that have a clearly defined discrete notion of work. Continuous applications, however, are different. They basically just stream data from a server to a client. There is not real discrete of work. However, continuous applications like movie players, are clearly an important class of applications for mobile users. As such, I plan to add support for these continuous applications. This will involve defining how the tactics for a continuous application should be defined. Once this is done, support for continuous applications will need to be added to Chroma.

## 4.3 Middleware Layer to Interface with Existing Service Discover Protocols

Chroma uses servers to remotely execute application code. However, first, Chroma has to find those servers. A number of service discovery protocols (SDPs) have been developed by various researchers. These include JINI [45], UPnP [26], and BlueTooth proximity detection [1]. Each of these protocols were created to work in a different environment. However, they all can be used to discover remote servers in the environment. For my thesis, I will not be developing any new SDP. However, I will be creating a middleware layer that interfaces with existing SDPs and provides a common interface to Chroma. This satisfies my goal of having a common view of service discovery from Chroma's perspective.

## 4.4 Using Extra Servers in Overprovisioned Environments in a Fair Manner

Each Chroma client is able to use extra servers in an overprovisioned environment to improve application performance. However, if Chroma clients automatically use all extra servers available, they will end up competing with each other and affecting their performance. A centralized scheme for determining which extra servers can be used by which Chroma client is not feasible in a mobile environment. Hence, I will be developing distributed algorithms that Chroma clients can use to ensure that each client is using the extra servers in a globally fair manner.

## 4.5 Optimizations Possible With Using Extra Server Resources

Currently I have only looked at three performance optimizations possible by using extra server resources. I plan to investigate other methods of using these resources that can also be used in a completely automatic fashion and quantify the performance benefits of these methods.

## 4.6 Integration with Prism

For Chroma to decide how to adapt any particular application, it needs to know what the user's preferences regarding that application are. In order to obtain these preferences, one of the goals is to integrate Chroma with Prism. Prism works at the task layer and keeps track of user preferences. As such, obtaining this information from Prism would be very beneficial to Chroma. However, designing the interface between Chroma and Prism is non trivial. How should information be exchanged between Chroma and Prism? What information is exchanged between Chroma and Prism? How does Prism inform Chroma of changes in user preferences? Can Prism obtain information from Chroma regarding various applications? If yes, how will this be done? These are some of the questions that need to be addressed before Prism can successfully be integrated with Chroma.

## 4.7 Making the Development of Adaptive Mobile Applications Easier

The preliminary 4-step process shows promise in making the development of adaptive mobile applications from legacy applications easier. Preliminary results of this 4-step process coupled with some case studies are in a CMU Tech Report [3]. However, more work still needs to be done in the following areas:

### 4.7.1 Support for Continuous Applications

Currently, Chroma doesn't support continuous applications. As such, the mechanisms designed to reduce the time to add applications to Chroma may not work for continuous applications. Once Chroma supports continuous applications, the effectiveness of the current mechanisms in supporting continuous applications will become known. The mechanisms will then be modified and/or extended to support continuous applications.

### 4.7.2 Creation of Mechanisms to Ease Server-side Code Development

The mechanisms to reduce the time needed to develop adaptive mobile applications currently only work for the client half of that application. However, any application can be remotely executed requires a server component. Currently, the server component has to be written from scratch. This is extremely time consuming and unscalable in the long run. I plan to work on developing mechanism which will reduce the time needed to develop the server component for mobile applications.

## 4.8 Validation

### 4.8.1 Metrics for Validation

Successfully evaluating this thesis first requires identifying metrics that can be used for validation. Some potential metrics include ”response time”, ”latency” and ”average load per server”. However, more work needs to be done to clearly identify the metrics used for verifying each part of this thesis.

### 4.8.2 Generality of Tactics

I need to show that tactics are a general enough concept to be useful for mobile computing applications. To validate this, I plan to define the tactics for a large number (about 5 to 10) of useful mobile applications. These applications will include both discrete and continuous applications.

### 4.8.3 Performance of Chroma

The effectiveness of Chroma will be demonstrated by showing that Chroma can provide good performance for a large number of applications (about 5 to 10). I already have tentative performance figures for Chroma for three mobile applications. Having, more applications are needed to verify that Chroma can provide good performance for a large number of applications. This performance study should also verify Chroma’s ability to automatically use extra resources to improve application performance.

### 4.8.4 Reducing the Time for Developing Adaptive Mobile Applications

A sizable portion of my thesis will involve developing software engineering methods for reducing the development time of adaptive mobile applications. Verify the effectiveness of the methods will not be easy. My current idea includes measuring the time needed to add applications to Chroma using the developed methods versus the time needed to add the same application to Chroma without the developed methods. I also plan to measure the time needed by other people to use my methods to add new applications to Chroma.

## 4.9 Open Problems

### 4.9.1 Automatic Methods of Abstracting Application Tactics

Currently, the application writer specifies the tactics of his application using a simple declarative language. Automatically determining the tactics of an application, without any application writer help, is a known hard problem. However, it is an open problem as to whether it would be possible to build a system that starts from a small set of tactics specified by the application writer and figures out any extra tactics required by monitoring the execution of the application.

# 5 Motivating Scenarios

This thesis will help to enable the following scenarios: The scenarios are listed along with some of the technology necessary to realize each scenario.

## 5.1 Scenario 1

*Tim is traveling in Europe with his iPAQ and he wants his iPAQ to perform language translation for him. The translation should always return in 1 second. If there are no remote servers available, Chroma will use the iPAQ to perform the translation. When Chroma discovers the presence of remote servers, it will seamlessly use them to do the translations. Using remote servers will result in better translations as the remote servers have access to larger vocabulary files. Chroma ensures that the latency constraint of 1 second is still satisfied. Tim is unaware of the decision making of Chroma.”*

This scenario requires Chroma to be able to do the following things:

- discover the presence of remote servers
- be able to adapt discrete applications
- seamlessly transfer execution of a program from the local machine to a remote server

- be able to do all this while satisfying user-specified latency goals

## 5.2 Scenario 2

*Jim wants to watch a video on his handheld device. However, the video itself is too big to store on the handheld device. Hence, it needs to be streamed in real time to the device. Chroma automatically sets up a connection to a video server and starts streaming the video on demand to Jim's handheld. However, in the middle of the video, the bandwidth from Jim's handheld to the video server drops. Chroma, automatically does two things. First, it starts using a lower quality video stream from the server to ensure that Jim's video is not interrupted. Second, Chroma decides if it would be better to switch Jim to another video server that has better bandwidth. If Chroma decides to switch Jim to another video server, it will have to ensure that the video feed that Jim receives is not interrupted in anyway. Jim may notice that Chroma is adapting the quality of the video stream but the quality that Jim receives should be acceptable to him and he should not have to intervene in any way."*

This scenario requires Chroma to be able to do the following things:

- discover the presence of remote servers
- be able to adapt continuous applications
- be able to manage multiple video streams at a time
- seamlessly transfer from one stream to another
- be able to do all this such that the user doesn't see any gap in the video stream

## 5.3 Scenario 3

*John is using his palm sized device to perform speech recognition. His palm sized device has no significant computational capabilities and must depend on remote servers to achieve decent performance. However, John is experiencing a high standard deviation in his latency as the remote server currently being used is also being used by other devices. Chroma notices John is in a smartroom and that there are a number of remote servers available. Each of the servers is being used to do different tasks. However, Chroma decides to parallelize the speech recognition by sending the same recognition task to multiple remote servers. The recognition that returns the fastest is given to John. By using multiple servers, Chroma maximises the probability that John's recognition will be performed as fast as possible as it is quite likely that one of the servers being used will be unloaded at the time the request is made.*

This scenario requires Chroma to be able to do the following things:

- discover the presence of remote servers
- be able to use more than one server at a time
- make intelligent decisions about how to use extra servers
- automatically parallelize applications such that the same operation can be done at multiple servers
- use intelligent resource management policies such that the extra servers are being used in a globally fair way

## 5.4 Scenario 4

*Joe is a new hire at the company. His manager proclaims him an "adaptation expert" and wants him to develop an adaptive video player for handheld devices running Linux and X. Not being a video expert, Joe collaborates with domain experts in order to understand how to make video players adaptive. As a result, Joe realizes that video playing adaptation is based on image quality and frame rate.*

*Joe decides to use the XAnim application as the base. He modifies XAnim to adjust its image quality and frame rate to the available resources. This is a tedious and iterative process as Joe has to make extensive changes to the XAnim source.*

*Finally, Joe achieves the desired modification: XAnim now uses an underlying resource management layer to adapt its behavior to the available resources. His manager is delighted and demands a similar modification of MediaPlayer, a video player for Windows CE. The functionality of MediaPlayer is similar to that of XAnim; so are the adaptive extensions to be added. However, the code base is completely different, and so is the underlying OS. Joe must redo all the painstaking work that he put into XAnim for MediaPlayer. Then he needs to start work on the five other video players that cover the Macintosh, Solaris, IRIX, BSD and Windows 2000 operating systems. His hell is just beginning!!*

*However, help is on hand for Joe. Using the software engineering methodology developed for Chroma, Joe is able to reuse the knowledge he learnt when modifying XAnim to quickly and easily modify the other applications. The tools provided by Chroma make it easy for Joe to easily target the applications to different operating systems. Finally, the methodology makes it easy for Joe to take an existing application (that Joe knows very little about) and quickly modify it to be an adaptive mobile application.*

This scenario requires the following tools and mechanisms:

- a simple-to-use, yet powerful, methodology for quickly making legacy applications adaptive and mobile
- a set of tools to facilitate and speedup this process

### 5.5 Scenario 5

*Jane is watching a football match on her iPAQ. Chroma detects that the bandwidth from the iPAQ to the video server is adequate for streaming the video at full quality with audio. As Jane moves around, the bandwidth from her iPAQ to the video server keeps changing. Chroma automatically changes the video stream (from full quality to medium quality etc.) to ensure that Jane's video stream is not interrupted. Jane notices that the quality of the video is changing but the feed is not interrupted and the quality provided by Chroma is always acceptable to Jane. However, Jane enters an area with minimal wireless connectivity. The bandwidth available to the video server drops below the minimum level required for Chroma to provide even the lowest quality video. Chroma reports that it is unable to satisfy any level of adequate performance given the current resource availability to the task layer. This task layer sits between Chroma and Jane and keeps track of Jane's global preferences regarding resources and applications. The task layer notices that the bandwidth available, while insufficient for streaming video, is adequate for retrieving just the audio stream of the live football game. The task layer then kills the previous streaming video application and then spawns a streaming audio application. It instructs Chroma to manage this application and to choose the correct audio bit-rate for the current bandwidth. Jane loses her video feed but the video feed is replaced with an audio feed. When the bandwidth becomes suitable for the video feed, the task layer will notice this and restart the video streaming application."*

This scenario shows the relationship between Chroma and the task layer. Chroma is responsible for adapting within an application. The task layer is responsible for choosing the correct applications, given the current resource availability, that will best satisfy the users requirements. Making this scenario a reality requires the following components

- a working task layer. For this, I plan to use Prism.
- development of the proper interfaces to pass information between an adaptation layer that works at an application layer (like Chroma) and an adaptation layer that works at the user/task layer (Prism).

## 6 Work Items

This section describes the completed and remaining work in this thesis, as an itemized list. The items followed by a single star are considered necessary for a minimum acceptable thesis; those followed by two stars are part of the expected thesis. Three stars mark the bonus items. Work already done is marked by a $+$, and items that I plan to leave to future researchers by $-$.

- Semantics and Language for Tactics
  - § Specification of the semantics of tactics in prose form$+$

  - § Initial language for describing tactics of discrete applications+
  - § Verification of language through multiple application case studies ⋆
  - § Extension/Modification of language to address requirements of applications ⋆
  - § Extension of language to support server side specifications ⋆⋆
  - § Extension of language to support continuous applications ⋆⋆

- Develop tactics-based remote execution system
  - § Basic Prototype of Chroma +
  - § Implement robust remote execution subsystem ⋆
  - § Develop middleware service discovery layer that leverages existing service discovery protocols ⋆
  - § Extend Chroma to support continuous applications ⋆⋆
  - § Add support for automatically parallelizing applications ⋆⋆
  - § Integration with Prism ⋆⋆⋆
  - § Ability to dynamically add/change/delete utility functions ⋆⋆⋆

- Verify effectiveness of tactics
  - § Initial case studies +
  - § Verify effectiveness with a large number of discrete applications ⋆
  - § Verify effectiveness for continuous applications ⋆⋆
  - § Develop a general tactics model from multiple case studies −

- Demonstrate extra performance benefits of tactics
  - § Initial validation of using extra servers because of tactics +
  - § Extend Chroma to automatically use extra servers without application intervention ⋆
  - § Develop mechanisms to ensure extra servers are used in a globally fair manner ⋆⋆

- Verify effectiveness of software engineering methods in reducing mobile application development time
  - § Develop initial methodology and tools to reduce development time +
  - § Verify methodology with multiple discrete applications ⋆
  - § Extend and verify methodology for continuous applications ⋆⋆
  - § Develop mechanisms to automatically generate server side code ⋆⋆⋆
  - § Extend methodology to generate necessary Prism interfaces and code ⋆⋆⋆

# 7 Related Work

## 7.1 Remote Execution Systems

There have been a number of application-aware remote execution systems such as Abacus [2], Coign [19] and Condor [5]. They perform well in environments where resource availability does not change between the time the system decides how to remotely execute an application and when it actually performs the remote execution.

However, this assumption comes under fire in mobile environments. These environments are characterized by highly variable resource conditions that change on the order of seconds [10, 14, 37]. Overcoming this uncertainty requires application-specific knowledge on how to remotely partition the application.

An extra benefit of acquiring this knowledge is that it allows us to utilize additional resources in over-provisioned environments such as smart spaces with many idle compute servers. It is envisioned that these environments will become increasingly common in the new future. Chroma is designed to opportunistically use these extra resources to improve application performance. I know of no other system that does this.

The largest contribution to the concept, design and implementation of my system comes from Odyssey [29] and Spectra [13]. Odyssey showed that it was important to degrade application quality as resources change in order to maximize user quality. It demonstrated the importance of adapting to network bandwidth in mobile scenarios, and proposed the first generic API for application-aware adaptation. Spectra provided remote execution capabilities to Odyssey and showed that it was possible to server battery power by performing remote execution.

There have been other systems that have looked at the problem at partitioning applications. These include systems that performed object migration like Emerald [20] and systems that performed process migration [11, 27, 30, 32]. For my thesis, I hope to avoid having to perform process and/or object migration. I am concentrating on the problem of identifying useful remote execution partitions of existing applications and I assume that the servers already have all the code necessary to execute the applications (I use a distributed file system to ensure this). Other systems [35] looked at the problem of service composition or the building of useful applications from components available in the environment. In my thesis, I will only address the problem of partitioning existing applications and not the problem of constructing useful applications from existing infrastructure.

### 7.2 Software Engineering Methods

In this work, I do not intend to develop any fundamentally new software engineering techniques but instead plan to reuse existing techniques like the concept of modularization first proposed by Parnas [33]. The technique of using stubs and a stub generator is derived from RPC [8]. RPC has shown the effectiveness of stubs in insulating system details from applications and the usefulness of a stub generator for automated code generation. We have simply applied these techniques to the realm of adaptation in pervasive computing.

The application description language addresses some of the same issues as 4GLs [24] and 'little languages" [6]. The latter are task-specific languages that allow developers to express higher level semantics without worrying about low level details. Our description language is similar as it allows application developers to specify the adaptation capabilities of their applications at a higher level without needing to worry about low level system integration details. Our stub generator converts this high level description into low level code for interfacing the application with the runtime. Another system that uses this method is CORBA [31, 43]. My system differs from CORBA in that it is primarily targeted towards mobile environments. CORBA has some support for mobile environments but it doesn't perform well in practice.

Other systems like aspect oriented programming [22] and Enterprise JavaBeans [36] also have features to enable the rapid prototyping of distributed applications. The solution that I am proposing is less heavyweight than any of these systems and is targeted explicitly towards mobile environments. It also requires substantial amounts of work to modify existing applications, that were not created for these systems, to use any of these systems.

Initial research [9] on adaptive multimedia applications concentrated on low-level system parameters, while concern for user-perceived quality attributes appeared later [25]. Expressing user satisfaction took an econometric slant, and new expressive power, with the introduction of utility functions in resource allocation systems in [34]. Capturing user goals and using that knowledge to drive systems is a cornerstone of recent work on expert systems that provide assistance to computer users. For example, Horvitz [17] uses Bayesian networks to perform inference on user goals and utility functions to evaluate the relative merit of alternative system actions.

## 8 Thesis Contributions

This thesis will show that it is possible to describe the remote execution capabilities of an application in a concise declarative form called `tactics`. I will show that it is possible to specify the tactics for a number of useful applications and that these tactics are independent of the underlying runtime environment. This allows tactics to be useful for other runtime systems and not just Chroma.

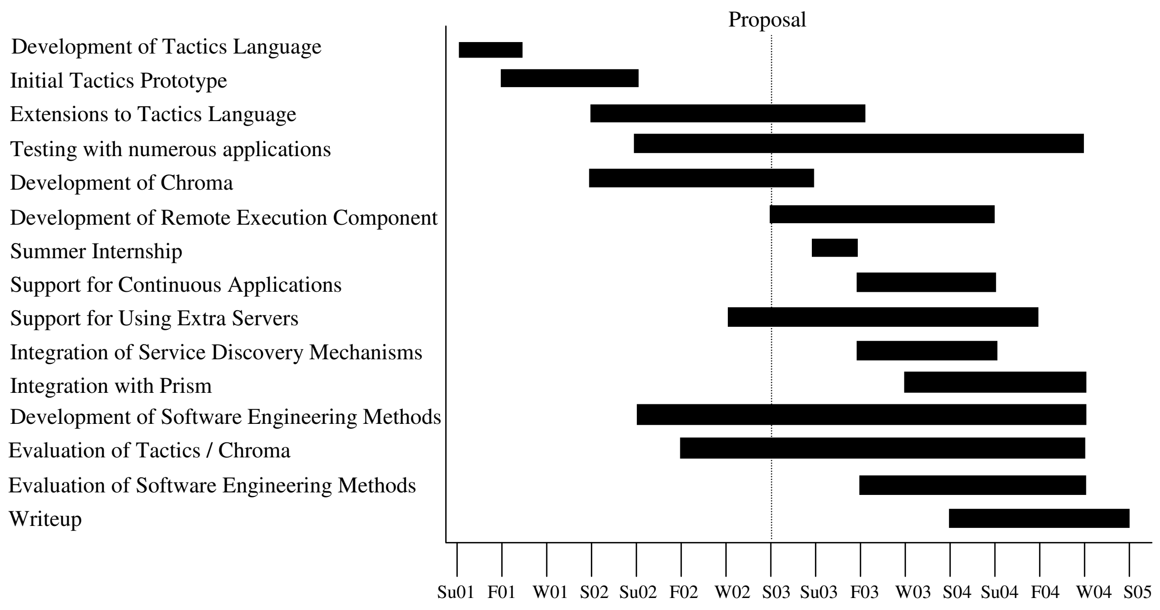
Based on this description, I will show that it is possible to build a powerful runtime system, Chroma, that uses tactics to provide good performance for computationally-intensive interactive applications. I plan to show that it is possible for Chroma to provide excellent application performance even though the resource management and adaptation algorithms used are simple. Developing algorithms that can accurately model the full dynamism of a mobile environments seems to be an impossible task. Given this, it is better to use simple algorithms that provide good performance in most of the cases than to use a complicated algorithm that provides better performance in some cases but is much harder to understand and breaks down at the corner cases. The use of simple algorithms that are still able to achieve good application performance, is an important achievement as it allows Chroma to be easily understood, easily developed and easily maintained.

Furthermore, I will show how Chroma can use tactics to automatically improve application performance in environments that have an overabundance of remote servers. This provides an important new capability for a remote execution system as overprovisioned environments are becoming increasingly common. When the user enters an overprovisioned environment, Chroma will automatically use these extra servers to make the performance of the users applications better. It does this by automatically parallelizing the users applications and executing them on multiple remote servers. By doing so, Chroma maximises the probability of the user being able to use a server that is completely unloaded and as a result, the user will experience the best possible performance. All these improvements happen automatically without any user intervention.

A key concern that arises when using extra servers is that one application may use all the available servers thus depriving other applications from using them. Since each Chroma client makes it own independent decision about which servers to use, it is necessary to develop a distributed algorithm that allows Chroma clients to choose the servers to use in a way that will ensure fairness. My goal is to develop various algorithms, compare their performance and show that it is possible for Chroma clients to achieve a fair use of available servers even though each Chroma client is making an independent decision.

This thesis will also demonstrate the benefits of coupling the task layer with the application/OS layer. For example, through the task layer, Prism, the user will be able to specify higher level constraints like ”make my battery life last 5 hours” or ”I only want to use 200 Mbytes of network bandwidth”. Prism will monitor these higher level user constraints and choose the correct applications to use based on these. It will then provide hints to Chroma explaining what the user’s preferences are. Chroma will use these hints to adapt the running applications in the way most suitable for the user. For example, Chroma may always choose a remote server if battery is a concern to the user, or Chroma may always perform local computation if network bandwidth is a concern to the user. Finally, this thesis will provide guidelines, mechanisms and a set of tools that will make the development of adaptive mobile applications easier.

## 9 Timeline



# A Language for Specifying Tactics

The tactics for each operation is specified in a file, named accordingly, which defines the list of execution units (keyword RPC) and the tactics (keyword DEFINE_TACTIC) for that operation.

A) Defining the execution units and their interfaces

This part of the grammar allows application writers to specify the various RPCs in their application that would be of interest to Chroma. The interface for the RPCs are also specified in this line.

Grammar

```
RPC <name> ( <parameters> ) ;
```

Explanation

```
RPC           :- keyword
<name>        :- name of this RPC.
<parameters>  :- interface of this RPC. This is expressed
                 using the grammar given below.

        <parameters>    = <var> | <var> , <parameters>
        <var>           = <mode> <type> <name>
        <mode>          = IN | OUT | INOUT
        <type>          = INT | CHAR | BYTE | FLOAT | DOUBLE | STRING |
                          user defined typedef
```

Example

```
RPC do_vq (IN STRING name, IN INT cpid, INOUT STRING Outfile, IN  BYTE flag);
RPC do_sq (IN STRING, IN INT cpid, INOUT STRING Outfile, IN  BYTE flag);
```

B) Specifying optional server groups

This part of the grammar allows application writers to specify optional server groups. This can be used to constrain a particular RPC in a tactic to only be executed at one of the servers specified in the group.

Grammar

```
SERVER <name> = <list of servers> ;
```

Explanation

```
SERVER        :- keyword
<name>        :- name of this server group

<list of servers>  :- the list of servers that make up this group.
```

Specifying this list of servers uses the following grammar

```
<list of servers>       =   SERVER  |  SERVER , <list of servers>
SERVER                  =  <name> | L   /* L = the local server                         */
                                        /* <name> = specific server name or IP address */
Example

SERVER local_and_gs129  = L, gs129.sp;
SERVER keyservers       = keyserver1, keyserver2, keyserver3;
SERVER trusted_hosts = moon.odyssey, 128.2.56.11;

C) Defining the tactics

This part of the grammar allows application writers to specify the ways in
which the various RPCs can be linked to one another in useful ways. I.e.,
the various tactics are defined here. The application writer can also
optionally specify specific servers and / or server groups to use for any
particular RPC within a tactic.

Grammar

DEFINE_TACTIC <name> = <list of RPCs> ;

Explanation

DEFINE_TACTIC :- keyword
<name>        :- name of this plan

<list of RPCs>  :- the list of RPCs which make up this plan. The application
writer can use curly brackets to group RPCs that should be run in parallel
if possible. The application writer can also optionally specify in each plan
whether individual RPCs in that plan should be run remotely, locally, at a
specific server, at any server within a server group or leave the decisions
up to the run-time.

Specifying this list of RPCs uses the following grammar

<list of RPCs>          =   RPC  |  RPC , <list of RPCs>
RPC                     =  <name> | <name> @ <constraint>  |  ( <list of RPCs> )
<constraint>            =  L | R | <server_name>   /* L = this must be performed locally
                                                   /* R = this must be performed remotely
                                                   /* <server_name> = use a specified server or server
                                                   /*  blank = runtime picks the server

Example

DEFINE_TACTIC local  =  do_vq@L,  do_sq@L ;
DEFINE_TACTIC hybrid = do_vq@keyservers,   (do_nq,  do_dq,  do_mq),  do_zq@R;
DEFINE_TACTIC remote = do_vq@R, (do_nq@gs129.sp,  do_dq@R,  do_mq@R),  do_zq@trusted_hosts;
```

## B Language for Specifying Adaptation Capabilities of Applications

```
Basic Information

This part of the grammar allows the application writer to specify the basic
```

identifying information about the application. These definitions are also used by Prism.

Grammar

```
Description      = <APPLICATION> <OPERATION> <REST>
<APPLICATION>    = APPLICATION app_name <TERMINATOR>
<OPERATION>      = OPERATION opp_name <TERMINATOR>
<TERMINATOR>     =  ;
```

Explanation

APPLICATION  & OPERATION are keywords.
app_name (STRING) is the name of the application.
opp_name (STRING) is the name of the operation.
The terminator ; is used to demarcate the end of the line

Example

```
APPLICATION janus;  OPERATION recognize;
```

Fidelity parameters

This part of the grammar allows the application writer to specify the variables of the application that are required to be known by Chroma for the purposes of calculating fidelities etc.

Grammar

```
<REST>                  = <FIDELITIES> | <REMOTE_EXECUTION> |
<FIDELITIES>            = <TYPEDEF> | <VARIABLE> |

<VARIABLE>              = <MODE> <TYPE>
<MODE>                  = IN | OUT | INOUT

<TYPE>                  = <INT>    | <CHAR>   | <BYTE> | <FLOAT> |
                          <DOUBLE> | <STRING> | <ENUM> | <TYPEDEF_NAME>
<INT>                   = INT var_name <ARRAY> <FROM> <TO> <DEFAULT> <TERMINATOR>
<CHAR>                  = CHAR var_name <ARRAY> <FROM> <TO> <DEFAULT> <TERMINATOR>
<BYTE>                  = BYTE var_name <ARRAY> <FROM> <TO> <DEFAULT> <TERMINATOR>
<FLOAT>                 = FLOAT var_name <ARRAY> <FROM> <TO> <DEFAULT> <TERMINATOR>
<DOUBLE>                = DOUBLE var_name <ARRAY> <FROM> <TO> <DEFAULT> <TERMINATOR>
<STRING>                = STRING var_name <ARRAY> <DEFAULT> <TERMINATOR>
<ENUM>                  = <ENUM_PARAMS> var_name <ARRAY> <DEFAULT> <TERMINATOR>
<ENUM_PARAMS>           = { <ENUM_VALUES> }
<ENUM_VALUES>           = enum_val , <ENUM_VALUES> | enum_val

<TYPEDEF_NAME>          = typedef_name  var_name <FROM> <TO> <DEFAULT> <TERMINATOR>

<ARRAY>                 = [ array_val ] |

<FROM>                  = FROM from_val |
<TO>                    = TO to_val |
<DEFAULT>               = DEFAULT default_val |

<TERMINATOR>            = ;
<TYPEDEF>               = TYPEDEF name <TYPE> <TERMINATOR>

Explanation

IN, OUT, INOUT, INT, CHAR, BYTE, FLOAT, DOUBLE, STRING, FROM, TO and DEFAULT
are all keywords.

var_name (STRING) is the name of the variable.
enum_val (STRING) is a particular value of the enum.
typedef_name (STRING) is the name of a previously declared typedef.
from_val (STRING) is the FROM value of a variable.
to_val (STRING) is the TO value of a variable.
default_val (STRING) is the DEFAULT value of a variable.
array_val (INT) is the number of elements in the array.
The terminator ; is used to demarcate the end of the line

Example

IN INT num_poly  FROM 0 TO 10 DEFAULT 5;
```

## References

[1] 3COM, Agere, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia and Toshiba. *Bluetooth Wireless Information Site*, 1999. http://www.bluetooth.com.

[2] Amiri, K., Petrou, D., Ganger, G., and Gibson, G. Dynamic function placement for data-intensive cluster computing. In *Proceedings of the USENIX 2000 Annual Technical Conference*, San Diego, CA, June 2000.

[3] Balan, R. K., ao Pedro Sousa, J., and Satyanarayanan, M. Meeting the software engineering challenges of adaptive mobile applications. Technical Report CMU-CS-03-111, Carnegie Mellon University, Pittsburgh, Pennsylvania, Feb. 2003.

[4] Balan, R. K., Satyanarayan, M., Park, S., and Okoshi, T. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, May 2003.

[5] Basney, J. and Livny, M. Improving goodput by co-scheduling CPU and network capacity. *Intl. Journal of High Performance Computing Applications*, 13(3), Fall 1999.

[6] Bentley, J. Little languages. *Communications of the ACM*, 29(8):711–21, 1986.

[7] Birrell, A. D. and Nelson, B. J. Implementing remote procedure calls. In *Proceedings of the ACM Symposium on Operating System Principles*, page 3, Bretton Woods, NH, Oct. 1983. Association for Computing Machinery, Association for Computing Machinery.

[8] Birrell, A. D. and Nelson, B. J. Implementing remote procedure call. *ACM Transactions on Computer Systems*, 2(1):39–59, Feb. 1984.

[9] Clark, D. D., Shenker, S., and Lixia, Z. Supporting real-time applications in an integrated services packet network; architecture and mechanism. *ACM SIGCOMM '92*, 22(4):14–26, aug 1992.

[10] D. Eckhardt, P. S. Measurement and analysis of the error characteristics of an in-building wireless network. In *Proceeding of ACM SIGCOMM*, pages 243–254, Stanford, California, October 1996.

[11] Douglis, F. and Ousterhout, J. Transparent process migration: Design alternatives and the Sprite approach. *Software Practice and Experience*, 21(7):1–27, July 1991.

[12] Flinn, J. and Satyanarayanan, M. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 48–63, Kiawah Island, SC, December 1999.

[13] Flinn, J. and Park, S. and Satyanarayanan, M. Balancing Performance, Energy, and Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on. Distributed Computing Systems*, Vienna, Austria, July 2002.

[14] Forman, G. and Zahorjan, J. Survey: The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

[15] Frederking, R. and Brown, R. D. The Pangloss-Lite machine translation system. In *Expanding MT Horizons: Proceedings of the Second Conference of the Association for Machine Translation in the Americas*, pages 268–272, Montreal, Canada, 1996.

[16] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P. Project Aura: Toward Distraction-free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, April-June 2002.

[17] Horvitz, E. Principles of mixed-initiative user interfaces. In *Proceedings of CHI '99, ACM SIGCHI Conference on Human Factors in Computing Systems*, Pittsburgh, PA, May 1999.

[18] Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In Cooper, G. F. and Moral, S., editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 256–265, San Francisco, July 24–26 1998. Morgan Kaufmann.

[19] Hunt, G. C. and Scott, M. L. The Coign automatic distributed partitioning system. In *Proceedings of the 3rd Symposium on Operating System Design and Implemetation (OSDI)*, pages 187–200, New Orleans, LA, Feb. 1999.

[20] Jul, E., Levy, H. M., Hutchinson, N. C., and Black, A. P. Fine-grained mobility in the emerald system. *ACM Transactions on Computer Systems*, 6(1):109–133, Feb 1988.

[21] Katz, R. H. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):611–17, 1994.

[22] Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. Aspect-oriented programming. In Akşit, M. and Matsuoka, S., editors, *ECOOP '97 — Object-Oriented Programming 11th European Conference, Jyväskylä, Finland*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. June 1997.

[23] Lee, C., Lehoczky, J., Siewiorek, D., Rajkumar, R., and Hansen, J. A scalable solution to the multi-resource QoS problem. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, pages 315–326, Phoenix, AZ, Dec. 1999.

[24] Martin, J. *Fourth-Generation Languages*, volume 1: Principles. Prentice-Hall, 1985.

[25] McCanne, S. and Jacobson, V. Vic: A flexible framework for packet video. In *ACM Multimedia*, pages 511–522, Nov. 1995.

[26] Microsoft Corporation. *Universal Plug and Play Forum*, June 1999. http://www.upnp.org.

[27] Milojicic, D. S., Douglis, F., Paindaveine, Y., Wheeler, R., and Zhou, S. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000.

[28] Narayanan, D. *Operating System Support for Mobile Interactive Applications*. PhD thesis, Carnegie Mellon University, Aug. 2002.

[29] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, pages 276–287, Saint-Malo, France, October 1997.

[30] Nuttall, M. A brief survey of systems providing process or object migration facilities. *Operating Systems Review*, 28(4), Oct. 1994.

[31] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 1999. Revision 2.3.1, ftp://ftp.omg.org/pub/docs/formal/99-10-07.ps.

[32] Ousterhout, J. K., Cherenson, A. R., Douglis, F., Nelson, M. N., and Welch, B. B. The Sprite network operating system. *Computer*, 21(2):23–36, Feb. 1988.

[33] Parnas, D. L. A technique for the specification of software modules with examples. *CACM*, 15(5):330–336, May 1972.

[34] Rajkumar, R., Lee, C., Lehoczky, J., and Siewiorek, D. Practical solutions for QoS-based resource allocation. In *The 19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 296–306, Dec. 1998.

[35] Raman, B. and Katz, R. An architecture for highly available wide-area service composition. *Computer Communications Journal, special issue on 'Recent Advances in Communication Networking'*, May 2003.

[36] Rutherford, M. J., Anderson, K., Carzaniga, A., Heimbigner, D., and Wolf, A. L. Reconfiguration in the enterprise JavaBean component model. In Bishop, J., editor, *CD 2002 — Component Deployment, IFIP/ACM Working Conference, Berlin, Germany*, volume 2370 of *Lecture Notes in Computer Science*, pages 67–81. Springer-Verlag, New York, NY, June 2002.

[37] Satyanarayanan, M. Fundamental challenges in mobile computing. In *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, May 1996.

[38] Satyanarayanan, M. Mobile Information Access. *IEEE Personal Communications*, 3(1), February 1996.

[39] Satyanarayanan, M. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17, Aug. 2001.

[40] Schneiderman, H. and Kanade, T. A statistical approach to 3d object detection applied to faces and cars. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 746–751, Hilton Head Island, South Carolina, June 2000.

[41] Sousa, J. and Garlan, D. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 29–43, Montreal, Canada, Aug. 2002.

[42] Sun Microsystems Inc. *Remote Method Invocation Specification*.

[43] Vinoski, S. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications*, 35(2):46–55, Feb. 1997.

[44] Waibel, A. Interactive translation of conversational speech. *IEEE Computer*, 29(7):41–48, July 1996.

[45] Waldo, J. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[46] Weiser, M. The computer for the twenty-first century. *Scientific American*, pages 94–101, September 1991.