RICE UNIVERSITY

User-Centric Framework for Device Aggregation

by

Rajnish Kumar

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

Willy Zwaenepoel
Dr. Willy Zwaenepoel, Chair
Karl F. Hasselman Professor of Computer Science

Alan Cox
Dr. Alan Cox, Associate Professor of Computer Science

Peter Druschel
Dr. Peter Druschel, Professor of Computer Science.

HOUSTON, TEXAS

MAY, 2003
USER-CENTRIC FRAMEWORK FOR DEVICE AGGREGATION

Rajnish Kumar

Thesis: Master of Science
Computer Science
Rice University, HOUSTON, TEXAS (MAY 2003)
Chapter 1

Abstract

As intelligent devices become affordable and wireless infrastructure becomes pervasive, the potential to combine, or aggregate, device functionality to provide a user with a better experience grows. Currently, the user must have a detailed understanding of the physical properties of the devices, the software services offered, and the dynamic behavior of the environment in order to form an appropriate aggregation. Even a small number of devices can be aggregated in many ways to perform a particular task effectively. This problem is more severe when the user is in an environment with unfamiliar devices, when a large number of devices are available, or when multiple tasks need to be performed simultaneously.

This thesis presents the design and implementation of a system for the user-centric aggregation of device functionality in a dynamic environment. It supports the automated selection of device functionality for aggregation using predefined descriptions of devices and their services. It facilitates the selection of the aggregation that best matches a user’s preferences using declarative policies, and it allows a user to express trade-offs between the quality of device attributes, user distraction, and aggregation stability. This approach enables a user to have a richer experience without having to constantly worry about device details and the aggregation of device functionality.
Acknowledgments

I thank my advisor Dr. Willy Zwaenepoel for his guidance throughout my research at Rice and reviewing this thesis. Part of thesis work was done at HP Labs during my summer internship, and I would like to acknowledge the contributions of Vahe Poladian, CMU to provide many valuable ideas and discussions, and doing the implementation jointly.

I would also like to thank Dr. Dejan Milojicic, Alan Messer and Ira Greenberg for introducing me to the idea of device aggregation, and providing the fruitful comments throughout the thesis development.

I am grateful to the other members of my research group: Dan Wallach, Eyal De Lara, Ping Tao who helped me develop the useful skills to do good research and helped me out in many small but important situations. I am also grateful to Amit Kumar Saha and Santashil Palchaudhary for their help in LaTex tips while writing the thesis.

I thank the other members of my thesis committee: Dr. Alan Cox and Dr. Peter Druschel for their suggestions which greatly improved the quality of this thesis.
## Contents

1 Abstract
   Acknowledgments
   List of Illustrations

2 Introduction

3 CAFE System Model
   3.1 Definitions
   3.2 System Assumptions
   3.3 Design Goals

4 CAFE System Architecture
   4.1 Major Activities
      4.1.1 Request Task
      4.1.2 Browse for All Devices
      4.1.3 Generate Aggregations
      4.1.4 Other Steps
   4.2 Scoring mechanism
      4.2.1 Device policies and device scoring
      4.2.2 Aggregation policies and aggregation scoring
      4.2.3 Stability and future planning
      4.2.4 Adaptation to change
      4.2.5 Adaptation to handle multiple user requests


4.2.6 Policy recommendation engine ..................................... 24

5 Implementation .......................................................... 25
   5.1 System Components and Initialization ............................. 25
   5.2 Service Description and Registration ............................. 26
       5.2.1 Task Request and Satisfaction ............................. 27
       5.2.2 Aggregation Scoring ......................................... 28
       5.2.3 Policy Suggestion ........................................... 30

6 Evaluation ...................................................................... 32
   6.1 Results ................................................................. 34
   6.2 Lessons Learned ..................................................... 36

7 Related Work .................................................................. 39

8 Conclusion ...................................................................... 41

Bibliography ..................................................................... 42
Illustrations

2.1 The number of possible aggregations increases combinatorially with an increase in the number of devices or tasks. The number of aggregation selections was computed by applying simple composition rules over the devices listed in the diagram. 4

4.1 The CAFE Architecture. 12
4.2 Steps involved in finding the best aggregation. 13
4.3 Scoring hierarchy. The final score of the aggregation is the weighted sum of the scores of the participating devices according to the aggregation policy. If this aggregation is the result of re-aggregation because of some change, the final score is adjusted to account for user distraction. This adjustment is applied in an ensemble-level policy. 17
4.4 An example of scoring sound devices using a service scoring policy for playing sound when quality and privacy are the attributes being considered. The numbers in the circles give the score for the device after applying the service policy. 18

6.1 Request Input Page. 33
6.2 User preferences form page. 34
6.3 Aggregation using policy suggestion engine. 35
6.4 CAFE's response time for handling one task. First two lines, 'Finding all aggregations' and 'Candidate Selection' show the corresponding response times for handling a new task. 'Device disappearance' line shows the response time for reaggregation in case of disappearance of a participating device from a running aggregation.  36

6.5 CAFE's response time for finding aggregations for two simultaneous tasks.  37
Chapter 2

Introduction

Mobile consumer electronic devices are becoming ubiquitous, and users now own several of these devices, such as a laptop, a PDA, a digital camera, and a smart phone. In addition, a user’s surroundings may contain fixed devices such as a desktop computer, speakers, a room projector, or a keyboard. Over the next few years, wireless devices will become available in the home, in the office, and in public places such as airport kiosks and coffee shops. Individually, these devices offer a wide range of computing capabilities. Combined together, however, they offer much greater functionality and can significantly enhance a user’s experience.

Imagine that Alice, a traveling consultant, visits one of her clients on a business trip. She has a laptop with built-in speakers, a PDA, and earphones. She also has access to a conference room during her visit that is equipped with a wall projector, a flat-screen monitor, a surround-sound stereo system, and a set of tabletop speakers. Alice wants to watch a corporate video that is stored on her laptop. One obvious choice is to watch the video entirely on the laptop, using it to decode the media stream, display the video, and play the sound. However, she prefers a large screen with sufficient color depth to ensure that similar colors can be differentiated. In addition, the presentation is confidential, so she would prefer to listen to the sound privately, but without compromising its quality. Lastly, because it is important for her to see the video, she would prefer that the selected devices remain available for the length of the video.
The available devices allow many possible aggregations* that satisfy her requirements. However, to select an aggregation that satisfies her preferences, she has to know the properties of each device, such as the size and color depth of the displays and the quality of each sound device. It is not obvious whether the projector is the best choice to display the clip because, although it has the largest screen available, its color depth is somewhat limited. Choosing the correct sound device presents similar challenges. In addition, Alice may have not noticed some available devices, such as the conference room speakers.

The complexity of selecting devices for aggregation increases as more interchangeable devices become available, leading to a combinatorial increase in possible solutions. In addition, if Alice would like to perform multiple tasks there will be a similar increase in complexity, as well as a need to arbitrate the allocation of devices between competing tasks. Figure 2.1 illustrates the selection complexity for five common devices (three with audio support, three with video displays, and three with text entry capabilities) being used for three common tasks.

Suppose that Alice has an intelligent device aggregation system running on her laptop. Instead of considering possible device combinations herself, Alice simply issues a task request to the system, selects some high-level policies about her preferences for relevant attributes like sound privacy, display quality, device stability, and minimum distraction, and the system determines which devices to aggregate to best match Alice’s preferences. Interestingly, the system suggests using the projector as the display even though it doesn’t have the best quality because the system has discovered that the laptop’s battery is low and will not last for the complete video. Alice accepts the system’s suggestion, and feels

*An aggregation is a group of devices that together perform a high-level user task. For example, a projector, ear-buds, and a PDA can be combined to form an aggregation for the purpose of playing an mpeg movie.
Figure 2.1: *The number of possible aggregations increases combinatorially with an increase in the number of devices or tasks. The number of aggregation selections was computed by applying simple composition rules over the devices listed in the diagram.*

relieved to know that she did not have to worry about the details of the device properties.

Now Alice is returning from her visit and she is at the airport. She realizes that she wants to access her email and communicate with a friend. She uses her PDA to handle these tasks. The aggregation system, also running on her PDA, discovers the closest devices that are available to be aggregated in the way that Alice likes. The system predicts Alice’s preferences based on a recorded history of her actions, and initializes the aggregation Alice would prefer. Comfortable with the familiar aggregation that the system selects, Alice feels at home even at the airport as she accomplishes her tasks.

This scenario presents the key functionality of an aggregation system for combining device functionality. Assuming that an infrastructure for aggregating device functionality becomes standardized, it highlights two important requirements that must be tackled to allow users to apply aggregation to their everyday tasks. First, devices need to be automat-
ically selected according to their functionality and attributes to provide a user with a richer experience. Second, there needs to be a flexible and intuitive mechanism to help a user express requirements. Combined together, these features allow a user to take advantage of a suitable aggregation that may not be obvious and may be complex to determine.

This thesis presents CAFE (Composition of Appliance Functionality in an Ensemble\textsuperscript{1}), a system that provides a solution to these identified requirements by employing a user-centric approach to automatically select the devices to aggregate. The system’s decisions are based on satisfying the user’s high-level, task-oriented preferences in various environments, where a combination of mobile and fixed devices are used to provide the user with a familiar experience. The user declaratively sets her high-level preferences for how CAFE should select devices.

CAFE has the following highlights.

- Declarative policies are used to capture user preferences about devices and aggregations as high-level abstractions. They allow the user to specify how to select an aggregation from several candidates, how to resolve resource conflicts among multiple user requests, and how to adapt an aggregation. This enables the user to focus on the desired experience instead of worrying about the details of the devices and the system.

- A metric called user distraction is used to compare candidate aggregations when performing reconfigurations. Mechanisms for specifying and quantifying distraction

\textsuperscript{1}An ensemble is a group of devices that can be accessed and controlled by the user. These devices are either owned by the user or borrowed temporarily, and can be considered available for the purpose of performing a user's tasks. An ensemble may have multiple aggregations instantiated for handling multiple user tasks.
have been identified. This information allows the user to establish a trade-off between the amount of distraction the user is willing to tolerate and the quality of the aggregation the user desires.

- A policy suggestion mechanism is employed to recommend policies to the user based on context and the user's aggregation history. This allows aggregation to be similar and predictable across different environments, and requires even less input from the user.

This thesis is organized as follows. Chapters 2 and 3 present the design goals and the architecture of the system. Chapter 4 describes the implementation. Results from an evaluation of the system and lessons learned are included in Chapter 5. Related work is described next. Chapter 7 presents some conclusions and suggests possible future work.
Chapter 3

CAFE System Model

In this section, we present CAFE’s design goals. This discussion is preceded by definitions of important terms and key assumptions.

3.1 Definitions

The following terms are used throughout the thesis.

**Distraction:** The inconvenience a user experiences when an aggregation is changed. For example, a user will experience some distraction if the display moves from the laptop to the wall projector, even if the quality of the display is improved. This inconvenience can distract the user from the user’s current task.

**Stability:** A metric that quantifies, as a percentage, how well a device will be able to perform a task to completion. For example, if a device’s battery is likely to be exhausted before the task is completed, then the device’s stability value is less than 100%. Similarly, a borrowed device that will probably have to be returned before the task is completed will also have a stability value that is less than 100%. We propose using the stability metric to discount a device’s quality score.

**Context:** Information that can be used to characterize an ensemble’s environment [8]. Examples include the user’s task, the ensemble’s devices, the user’s location, and the time of day.

**Policy:** A group of numerical weights and a high-level description that encode a user’s
preferences for various device attributes, types of devices, and types of tasks. An example of a policy would be: “Prefer a Large Screen and Prefer a Flat Screen,” which encodes a preference that gives a high weight to display devices that have a large, flat screen.

3.2 System Assumptions

Device and Service Model

Devices provide functionality that can be combined with the functionality of other devices in an ensemble. Devices are described in terms of the functionality they offer. Every device has a representative process that is responsible for announcing the device’s availability and functionality, and for providing information about the device’s dynamic properties. This representative can execute on the device itself, or on some other device.

Devices are assumed to be under the user’s complete control. They are either owned by the user, or borrowed for a given amount of time. CAFE does not allocate devices among competing requests made by multiple users. (This problem was considered in Rascal [13].)

Application Service Model

CAFE is designed to handle user requests that can profit from the aggregation of devices in an ensemble. We assume that applications are component-based, that they expose interfaces for remote activation, and that they can be easily integrated with other components or devices. Application components are remotely activated by the system to initialize an aggregation. The interfaces should clearly define the signatures of the activation methods, and the interface definitions should be specified in a device-independent language.

An application’s control logic should be decoupled from its input and output devices. This will allow the application to work with other devices in the ensemble transparently, without any user intervention. To support this decoupling, application requirements for
external devices should be expressed in a language that can be understood by other devices and the system.

**Network Model**

We assume that all devices are connected with a shared, wireless network. The environment may be quite dynamic, i.e. the set of devices that comprise an ensemble can change frequently. For example, devices can be turned on and off, borrowed or returned, or they can run out of power, lose network connectivity, or break.

**Infrastructure Support**

We assume the existence of middleware that allows applications to be remotely controlled on devices (e.g., started, paused, and stopped), similar to the functionality provided by middleware like Metaglue [7]. CAFE will determine the aggregation that is closest to a user's preferences for a requested task, and will use the middleware to instantiate the aggregation. CAFE will not manage the flow of data or control among the participating devices once the task is running.

### 3.3 Design Goals

CAFE provides support for the automatic selection of a device aggregation that satisfies a user's preferences. Here, we discuss the design goals for these objectives.

**Goal:** Allow the automatic selection of devices for aggregation in a way that best matches a user's preferences.

**Design:** To allow automatic aggregation, each kind of functionality provided by a device is viewed as a service, and a device is described as the services it can handle and the services it will need. A service is represented as a data-action pair. This approach is similar to the way MIME types are used in Internet messaging. Data types are specified using simple file
types such as ‘MPEG’ or ‘mp3,’ and actions are specified with simple descriptions such as ‘edit’ or ‘play’. This approach is simple and allows devices to be grouped into types.

We propose a simple mechanism to obtain user preferences, requiring little input from the user. It is difficult to obtain user preferences, and a good solution must balance accuracy against the amount of input required from the user. We propose using declarative policies that are user friendly and encode numerical weights for various properties. A user then only has to choose among policies, which have user-friendly names and hide (encode) the weights for the various properties. The policies are organized into a hierarchy, which further simplifies preference specification for the user.

Goal: Minimize user distraction.

Design: User distraction is minimized by employing stability and distraction metrics in the selection of devices and aggregations. When selecting an aggregation, the stability of candidate devices is quantified, and the user is allowed to balance stability against quality to minimize the possibility of a costly re-aggregation. The composition of an ensemble can change for predictable reasons such as the consumption of battery power or the end of a borrowing agreement. These factors should be considered when selecting an aggregation so a user can specify how important it is to have a stable working environment for the duration of a task.

Sometimes the user will determine that the gain in quality is large enough to outweigh the loss of stability. When forced to re-aggregate, changes that are potentially disruptive to the user are quantified and penalized, and the user is allowed to specify trade-offs between aggregation quality and distraction. Once a user is engaged in a task, it is potentially disruptive to switch a device that is being used, and that it should be possible to penalize such switches. The penalties should depend on the type of device that is switched and the type of task that is running. Further, the user should be able to specify how the penalties relate
to the quality scores. If a device that is part of a running aggregation becomes unavailable, re-aggregate must occur, and it must be done in a way that minimizes distraction. If a new device becomes available that has better quality than a device in a running aggregation, re-aggregation will occur only if the quality improvement outweighs the distraction penalty, as specified by the user.

**Goal:** Ensure acceptable system response time.

**Design:** The overhead of the system itself should be acceptable to the user for the system to be useful. Ensuring low response time presents challenges on resource-constrained platforms such as PDAs. Based on past work [12], we anticipate that handling requests for multiple tasks can lead to performance bottlenecks. We propose using optimized heuristics (described in section 4.2.5) to handle the case where the user issues the second request while an aggregation is already running for the first request.
Chapter 4

CAFE System Architecture

In this chapter, we present the CAFE architecture. We explain the mechanisms used to calculate the scores for candidate aggregations and to select the aggregation that best matches a user’s preferences.

The CAFE architecture is shown in Figure 4.1. The entire CAFE system runs on a selected device that we call the Coordinator. Any device in an ensemble can act as the Coordinator, which is responsible for running the five components shown in Figure 4.1. The shaded boxes represent the run-time components that we developed, and the components inside the dotted box represent the entire CAFE system. The main components of CAFE are:

1. **SLP Provider**: registers and discovers services using Service Lookup Protocol (SLP) [14]

2. **User Interface**: provides a web-based front-end for requests

![Figure 4.1: The CAFE Architecture.](image)
3. **Aggregator**: calculates all possible aggregations for a given user task(s)

4. **Candidate Selector**: selects the aggregation among the candidates that best matches the user’s preferences

5. **Policy Recommendation Engine**: predicts the user’s preferences based on the user’s past interactions with the system

### 4.1 Major Activities

Figure 4.2 presents the high-level flow of CAFE including interactions with the user and major computation steps. We now discuss these major activities.

#### 4.1.1 Request Task

We have developed a simple, Web-based interface for user’s to enter task requests. A user requests a task by specifying a data-action directive. For example, to play the movie “The Matrix,” the user would specify the data source “TheMatrix.mpeg” and the action directive “play.” The user can guide the system in the selection of a suitable aggregation by
identifying specific devices or preferences, but this is not always necessary because CAFE can predict the user's preferences based on the user's preference and execution history.

4.1.2 Browse for All Devices

SLP is used for device registration. Currently, a device registers its functionality with the Coordinator by posting its service description file and its length of availability. The coordinator then registers this information with SLP on behalf of the device. We chose this approach so that CAFE could be notified of registrations and de-registrations. We use a simple XML format to describe the services offered by the device, the attributes of the device, and the parameters required for service execution.

A device is described in terms of the functionality it supports, in particular by a data-action directive. For example, a device that can play an mpeg movie is described as being able to handle the "play, mpeg" pair, that is, it provides the play-an-mpeg-movie service. Further, for each service that a device supports, the services that it needs are also specified. For example, a device that provides the play-an-mpeg-movie service may in turn need a sound service and a video service. Using this tree of service information, CAFE can automatically generate aggregations.

Device descriptions also contain the values of various attributes. We say that devices are of the same type if they support the same data-action directive. Devices of the same type have the same set of attributes.

Further, device descriptions contain any necessary information to perform an invocation, such as the handle of the executable that needs to be executed.
4.1.3 Generate Aggregations

The Aggregator module of the Coordinator is responsible for generating all of the candidate aggregations that can satisfy a user request. Aggregations are automatically generated by CAFE using the existing device descriptions. The data-action directive in the user's request is taken as an unsatisfied need, and expanded. Expansion of a candidate aggregation is accomplished by adding a device that is able to handle an unsatisfied need in the current candidate. As new devices are added to the aggregation, some needs become satisfied, but new needs may also be added. Expansion of the candidate stops when all of the needs are satisfied. At this point, the candidate aggregation is complete, or final. If we run out of devices before the candidate is final, then the request cannot be satisfied.

The core of the Aggregator module is the Java Expert System Shell (JESS), a rule-based engine [11]. Device functionality is expressed as facts, and rules are used to allow the aggregation of compatible devices. As mentioned earlier, a device functionality description contain needs, which are used to expand a candidate aggregation. A user's request is also described as a fact that needs to be satisfied. Asserting the user's request starts the chaining of rules. This process ultimately completes with the generation of a set of candidate aggregations that completely satisfy the request.

4.1.4 Other Steps

Once the Aggregator generates all candidate aggregations, the Selector module is invoked to rank the aggregations according to the user's preferences. User preferences are captured by high-level, declarative policies that describe the device attributes abstractly. Section 4.2 describes the policy mechanism that is used to rank the aggregations. Once the aggregations are ranked according to user preference, they can be displayed to the user so that the user
can select one, or the system can automatically instantiate the best candidate.

4.2 Scoring mechanism

CAFE uses multi-attribute utility theory (MAUT) [10] as the scoring mechanism to capture user preferences and rank the aggregations. MAUT is an established technique in utility theory and it has been used to model user preferences for product customization [6]. According to MAUT, the overall evaluation of a product can be defined as a weighted sum of its evaluation with respect to its relevant value dimensions (attributes) [20]. This simple mechanism can be applied to evaluate the aggregations as well. Apart from device properties, CAFE uses device stability and user distraction as additional value dimensions for evaluating the aggregations.

Because the number of relevant attributes can be very large even for a small ensemble, having a user provide the attribute values and weights may not be feasible. CAFE uses declarative policies to hide these details from users. Different policies encode different scores or weights for the device attributes and ensemble properties. A user is presented a list of such policies from which to select. It is easier for a user to think in terms of high-level policies like "large display for movie" than in terms of low-level attribute values.

CAFE employs a hierarchy of policies, which further simplifies a user's task of communicating preferences to the system. There are three policy levels: device-level policies that capture information about device attributes, aggregation-level policies that capture information about how devices are scored relative to each other, and ensemble-level policies that capture less tangible information such as aggregation stability, user distraction, and multiple-task tradeoffs.

Figure 4.3 gives a high-level picture of the scoring mechanism. A device is scored according to the values of its attributes and the user's preferences. An aggregation is scored
Figure 4.3: Scoring hierarchy. The final score of the aggregation is the weighted sum of the scores of the participating devices according to the aggregation policy. If this aggregation is the result of re-aggregation because of some change, the final score is adjusted to account for user distraction. This adjustment is applied in an ensemble-level policy.

by combining the scores of its participating devices and the user’s preferences. An ensemble is scored by combining the scores of the aggregations and less tangible metrics such as user distraction. The details of scoring are described below.

4.2.1 Device policies and device scoring

Device-level policies allow devices to be scored and permit devices that offer the same type of service to be compared. Examples of devices that offer the same type of service are a room speaker, a desktop speaker, and an earphone. For each type of device, a set of relevant attributes were identified that represents the device’s utility. For example, for sound devices, the selected attributes are the quality of the sound, whether or not the sound can be heard in privacy, whether or not a speaker has sub-woofers, and whether or not surround sound is supported.

We have identified a number of discrete values for each attribute at a coarse level of granularity. For example, for sound quality five values were selected ranging from “low” to “very high.” For each value, a score was assigned between zero and one hundred. We
Figure 4.4: An example of scoring sound devices using a service scoring policy for playing sound when quality and privacy are the attributes being considered. The numbers in the circles give the score for the device after applying the service policy.

use common sense when assigning scores to the values. For example, when scoring sound quality, higher is considered better. When scoring a binary attribute, the availability of the attribute is considered better than its absence.

A device scoring policy specifies a weight between zero and one for each of the key attributes of a device. These weights are normalized to add up to one. The score of a given device D according to policy P is computed as the dot product of the vector weights specified by the policy with the vector of scores for the device’s attributes. Applying MAUT, the device score is computed as:

$$DS(D, DP) = \sum_{i=1}^{d} aw_i(DP) * D(v_i)$$  \hspace{1cm} (4.1)$$

where $DS$ is the overall score of device $D$ according to device scoring policy $DP$, $d$ is the number of attributes for the type of device, $aw_i(DP)$ is the weight of attribute $i$ according to policy $DP$, and $D(v_i)$ is the score for the device’s value ($v_i$) for attribute $i$. Figure 4.4 gives an example of applying device scoring policies to compare sound devices. Note that the device that is considered best depends on the policy used. When sound quality is preferred, the room speaker is best; when privacy is most important, the earphones are best; and when both privacy and quality are equally preferred, the desktop speaker is best.
4.2.2 Aggregation policies and aggregation scoring

Aggregation-level policies are used to indicate that some devices are more important to the user than others when forming an aggregation for a particular user request in a particular user context. This is accomplished by having the policies provide weights to the devices. For example, when watching an action movie, a user may want a very good display and may be much less interested in the sound quality. On the other hand, when watching a music album, the user may be more interested in the sound quality than the quality of the display. Similar to device-level policies, aggregation-level policies are described using high-level names so that the user does not have to worry about numbers. Several aggregation-level policies are provided with the system for common tasks.

An aggregation score is computed as:

$$AS(A, AP) = \sum_{i=1}^{n} sw_i(D, AP) \times e(D_i) \times DS_i(D, DP_i)$$  (4.2)

where $A$ is the aggregation, $AP$ is the aggregation policy to be applied, $AS$ is the aggregation score, $n$ is the number of devices that are included in the scoring, $sw_i$ is the weight assigned to the device of type $i$ according to aggregation policy $AP$, and $DS_i$ is the score that the device of type $i$ (in this case $D$) gets when scored using formula (1). $e(D_i)$ is a percentage indicating the availability of device $D_i$. We will discuss this metric further in the next chapter.

4.2.3 Stability and future planning

Once an aggregation is instantiated, changing it may inconvenience the user. Therefore, it might be worthwhile to sacrifice the quality of the aggregation to minimize the probability of change. For example, if a device uses a resource such as bandwidth near its capacity, then it might be better to choose a different device so that unexpected variations do not
necessitate re-aggregation. Similarly, if a device is expected to become unavailable before
a task is completed, say because its lease will end, then it might be better to start with a
different device.

While we do not compute a stability metric in the current system, one can computed
it in several ways. One approach is to collect historical information about the availability
of devices, and to use this information to predict future availability. Another approach is
to determine the expected length of the task, say from the mpeg file, from the advertised
length of the conference, or by asking the user, and then compute availability accordingly.

In formula 4.2, $e(D_i)$ is the term that accounts for stability. It expresses the probability
that the device will be available throughout the duration of the task. Intuitively, this weight
reduces the contribution of the device score by some percentage that is equal to the prob-
ability of that device becoming unavailable. We currently multiply the quality score of a
device by a factor proportional to its stability probability, and use this value when scoring
the device in an aggregation. This makes the stability of the devices propagate to the ag-
gregation level. In this way, we can also think of stability at the aggregation level where it
is the probability that an aggregation will be able to perform a task for the desired amount
of time.

4.2.4 Adaptation to change

Although CAFE plans for the future and attempts to minimize the effect of changes, an
ensemble is expected to be dynamic enough to require an efficient way to adapt existing
aggregations. An aggregation may need to change because one of its devices becomes un-
available, because a new device becomes available, or because the user requests a new task.
Some of the new aggregations may provide better quality or may better satisfy the user's
preferences. When ranking new aggregation candidates, penalties are used to account for
any inconvenience that the user incurs when an existing aggregation changes. This type of inconvenience creates distraction for the user.

Note the relationship between stability and distraction. Stability is considered when an aggregation is created, and represents the probability that the aggregation will have to change. Distraction is considered when an event occurs that may lead to re-aggregation. Also, note that re-aggregation may not actually occur because the distraction penalty may exceed the benefit of increased quality.

Minimizing a user’s inconvenience is an important factor to consider when adapting an aggregation. For example, a user may want to avoid having the display move, or a user may want to avoid moving to a different microphone. On the other hand, moving the mpeg decoder from one computer to another may not significantly bother the user. Changing some devices may cause more inconvenience to the user than changing others. In particular, devices that directly interact with the user present the highest potential for inconvenience. Further, the extent of the inconvenience depends on the kind of task being performed. For example, when watching a movie, the inconvenience associated with changing the display device is probably more severe than the inconvenience of changing the sound device.

User inconvenience is computed as a penalty score that quantifies the amount of inconvenience that the user will incur. We call this the aggregation difference penalty. The formula for computing this measure is:

\[ ADS(A', A) = \sum_{i=1}^{d} b_i(S'_i, S_i) * DDP_i \]  

(4.3)

where the sum is taken over all of the devices that have non-zero penalties, \( b_i \) equals zero if \( D'_i \) and \( D_i \) are the same device and one otherwise, and \( DDP_i \) is the penalty score for switching the \( ith \) device type. The sum of all \( DDP \) scores is calibrated to add to 100.

This difference penalty captures the amount of inconvenience that the user will incur if the device change occurs. In some situations, the user will prefer to minimize the amount
of change at the expense of quality. In other situations, the user will prefer better quality at the expense of change. Thus, it is intuitive to allow the user to specify trade-offs between quality and change.

We propose a re-aggregation mechanism that considers a user's tolerance for changing the current aggregation. We define policies that specify varying trade-offs between distraction and aggregation quality. These policies are declarative and can be selected by the user, or they can be selected automatically by the system based on the context. They are applied when an event occurs that makes a change possible.

A user's tolerance for change is represented by weights for distraction penalties and aggregation quality. It is calculated by the following formula.

\[ ES_R(A') = AS(A', AP) - q_d \times ADS(A, A') \]  \hspace{1cm} (4.4)

where \( AS \) is the aggregation score for \( A' \) according to aggregation policy \( AP \), \( q_d \) is a weight between 0 and 1 representing inconvenience, and \( ADS(A, A') \) is the difference between the scores of the new candidate aggregation \( A' \) and the currently running aggregation \( A \).

As an example, when the user chooses a policy to emphasize quality, \( q_d \) is zero. On the other hand, if the user wants to minimize distraction, \( q_d \) will have a non-zero value based on the user-selected policy.

### 4.2.5 Adaptation to handle multiple user requests

Sometimes the user wants to execute multiple tasks simultaneously. When this occurs, the system needs to arbitrate the use of devices among the tasks.

To support this arbitration, we allow users to specify the importance of the tasks relative to each other. This preference is captured as ensemble-level policy, and it is applied to
compute the ensemble-wide score as the weighted-sum of the aggregation scores using weights from the preferred policy.

The formula for computing the ensemble-wide score is:

\[ ES_R(EP, A_1, A_2) = q_1 \cdot AS(A_1) + (1 - q_1) \cdot AS(A_1) \]  (4.5)

where \( EP \), the ensemble policy for assigning weights to aggregations, \( AS(A_1) \) is the aggregation score for candidate aggregation \( A_1 \), \( AS(A_2) \) is the aggregation score for candidate aggregation \( A_2 \), \( q_1 \) is the weight assigned to the first task, and \( q_2 \) is the weight assigned to the second task.

Two approaches were used to generate suitable aggregations for two tasks. The first approach is naive, and enumerates all possible aggregation pairs using nested loops. It is possible to slightly optimize this approach by running the outer loop for the more favored aggregation. Nevertheless, finding the best aggregations that match the user’s preferences requires complete enumeration of all candidates.

In the second approach, which we call the greedy heuristic, the JESS engine is used to generate aggregation templates for each task. These aggregation templates contain the types of services that are needed to satisfy a particular type of request. Next, the services that are needed by both tasks are identified. Arbitration is needed only for these types of services.

Note that the formulae involved in calculating the device, aggregation, and ensemble scores are linear. Using this observation, the potential contribution of a given device to the final ensemble-wide score can be computed provided that the device is used in an aggregation to satisfy a particular request (e.g., first request, second request, and so forth). Consequently, we can arbitrate between devices of the same type for different requests by computing each device’s contribution, and then by allocating that device to the task for which it contributes most. This calculation takes considerably less time than the naive
4.2.6 Policy recommendation engine

The policy recommendation engine enables CAFE to predict a user’s preferred policies for the device, aggregation, and ensemble levels. This allows CAFE to aggregate devices even when the user does not specify policies. The policy engine selects policies that represent the user’s preferences based on the choices that the user has made in the past, the task being requested, and other aspects of the current context such as the time of day.

A decision tree algorithm is used to predict the policies that the user would select. A user’s past preferences and contexts are stored in history files, and these files are used as the learning set by the decision tree algorithms. The context contains the factors that may influence a user’s choice of policies, such as the user task or location. The history files are updated whenever the user provides preferences manually. Initially, the history files may not be rich enough to enable the decision tree algorithm to make accurate predictions. When this occurs, a pre-specified default policy is used for each of the three policy levels. This will be discussed further in the implementation chapter.
Chapter 5

Implementation

In this chapter, we will describe the implementation of CAFE’s components and the lessons we learned from the implementation. CAFE is implemented in Java and runs within a Java servlet engine. Its interface is Web-based and can be used from any device within an ensemble. The coordinator that hosts the CAFE components is discoverable, which makes the system more flexible in ad-hoc environments.

5.1 System Components and Initialization

The bulk of CAFE is composed of Java code, XML configuration files, and a JESS template file. Upon initialization, CAFE loads the JESS file and initializes the JESS engine. It then loads the XML configuration files and prepares to accept user requests.

The JESS template file contains definitions for fact templates and several rules for expanding aggregations. There are templates for requests, services, aggregations, and final aggregations. These templates will be described further below.

All of the XML files are kept in a repository (see figure 4.1). These files include service descriptions, policies, and user history information.
5.2 Service Description and Registration

A device registers its availability by executing an HTTP POST of its service description XML file to the registration URL. A service description XML file specifies the data type and action directive pair that the service can handle, the services required to support the data-action pair, values for attributes that are appropriate for that service type, and so forth. Here is a sample service description file.

```
<service name="MPEG Splitter" uniqueId="MPEGPlayerSplitter">
  <handles mime="mpeg" action="play"/>
  <virtualLocation>polian.hpl.hp.com</virtualLocation>
  <executableHandle>splitter.bat</executableHandle>
  <requires>
    <serviceReq mime="mp3" action="play"/>
    <serviceReq mime="avi" action="play"/>
  </requires>
</service>
```

The SLP component executes an XSL transformations [3] on a service description file to generate registration information for SLP. Because the format of this file is simple, it is possible to develop custom format translators that do not use XSLT. Employing custom translators would make it possible to decrease the footprint of CAFE because the XSLT libraries are memory intensive.

The SLP registration string for the previously described service is:

```
service:play.mpeg:MPEGPlayerSplitter://poladian.
  hpl.hp.com/(uniqueId=MPEGPlayerSplitter)
```

This is a "service:" URL that conforms to the SLP standard. Note that the abstract service type portion of the URL is used to specify the action-data pair that the service handles.
A device uses the registration string to register its services with the SLP system. Service registration remains valid for a fixed interval, after which the service registration expires. Devices are expected to re-register themselves with the SLP directory server within this expiration interval. This is enforced to ensure that the information at the coordinator is up to date. The length of the expiration interval is configurable and depends on how volatile the environment is. The disadvantage of enforcing an expiration interval is that the devices are forced to periodically re-register.

5.2.1 Task Request and Satisfaction

A user requests a task by specifying a data file and action directive pair. The extension of the data file is used as the mime type of the request. When the request is received, the aggregator component performs the following steps.

- Browse all of the services in the SLP system
- Generate a JESS fact for each service
- Enter each fact into the JESS engine
- Assert a fact that corresponds to the request
- Execute the JESS engine to generate aggregations
- Score the aggregations

Browsing the services in SLP is accomplished in two steps. First, a query is issued to identify all of the available service types. Next, for each service type, a query is issued to identify all of the services of that type. This allows all of the registered services to be browsed without any prior information. Initially, SLP was queried each time, but later this was changed to improve performance by caching the SLP information in memory using a Hashtable object. Browsing the SLP directory is slow because the API makes multicast network requests, which can take a long time to complete.
Once all of the registered services are obtained, XSLT transformations are used to convert the service descriptions into JESS facts. Here is the JESS fact corresponding to the previously mentioned service.

```
(assert
  (service (serviceId MPEGPlayerSplitter)
    (handlesDirective play_mpeg)
    (final no)
    (requires (create play_mp3 play_avr ))
))
```

When the JESS engine is executed, rules are fired based on matches among existing facts. When a rule is fired, new facts are generated. Initially, the fact corresponding to the request is matched with an appropriate service, and partial aggregations are generated. Partial aggregations are further expanded using facts that match the needs of the services in these aggregations. Aggregation expansion stops when all of the needs are satisfied, or when there are no more matches. As new aggregations are generated, we assert them as facts.

This approach allows every aggregation, partial or complete, to be stored in JESS during the generation process. This makes it possible to later query all of the complete aggregations. The design of the aggregation template also makes it possible to store information about the structure of the aggregation, including the dependency graph. This approach also supports "pinning" (allowing the user to specify a required device).

The JESS engine stops when no rules can be applied. At that point, a query is made for facts of type "final-aggregations." If none are found, then no feasible aggregation exists that satisfies the request. Otherwise, the list of final aggregations is obtained.

### 5.2.2 Aggregation Scoring

To score aggregations, CAFE needs to know 1) how to score each participating device, 2) how to assign weights to the devices in the aggregation, and 3) whether to apply an
ensemble-wide policy to account for the distraction caused by a re-aggregation.

For each type of service, attributes are identified that are common across all of the devices that support that service type. For each of these identified attributes, common sense is used to score all of its possible values. Here is the service scorer XML file for the display_video service.

```xml
<serviceScorer action="display" mime="video">
<attributes>
  <attribute name="size">
    <point value="xLarge" score="100"/>
    <point value="large" score="90"/>
    <point value="medium" score="80"/>
    <point value="small" score="30"/>
    <point value="verySmall" score="10"/>
  </attribute>
  <attribute name="flatScreen"/>
  ...
</attributes>
</serviceScorer>
```

A service scoring policy is a vector of weights that is applied to the attribute scores of a particular service to obtain a single score for that service. A service is scored by calculating a dot product between the weights vector and the scores vector of the service. Different policies allow different attributes to be favored. For example, a policy named “Large_Display” is weighted to favor the size of the display, and one named “Private_Display” is weighted to favor privacy.

Note that our approach for scoring attributes allows the user to assign arbitrary weights to the different device attributes. These weights should sum to one so that the device scores can be normalized across the ensemble. These policies allow a user to weight some attributes more than others and to add flexibility to the CAFE system, without requiring the user to provide a significant amount of input.

An aggregation scoring policy is implemented as a vector of weights that is applied to the individual service scores to obtain a single quality score for an aggregation. The
aggregation score is computed by applying the aggregation scoring formula (2).

5.2.3 Policy Suggestion

We used the implementation of the decision tree algorithm provided by WEKA [2] for policy suggestion. WEKA is a collection of machine learning algorithms for solving real-world data-mining problems. Although learning in our system is based on a decision tree, other data-mining approaches can be used with few changes to the implementation.

A data set file stores a user’s history, and is used as an input to the decision tree algorithm. It contains entries for context information and the policy selected by the user in the past for that context. The data set is stored in attribute-relation file format (ARFF), the format expected by WEKA. Here is an example of an ARFF file for predicting a device-level policy for the “display_video” task.

```arff
@relation policyEngine
@attribute task {play_mpeg, play_avl, run_email}
@attribute time {morning, midday, evening, night}
@attribute policy
    {Policy_Larger_Better, Policy_Private}
@data
    play_mpeg, night, home, Policy_Private
    play_mpeg, midDay, office, Policy_Private
    play_mpeg, midDay, conferenceRoom,
        Policy_Larger_Better
```

Separate data set files exist for each policy level that needs to have a policy selected. For device-level policies, there is a data set file for each service type supported by the end devices, such as display_video. Similarly, the system stores history information about aggregation- and ensemble-level policies in separate files.

The data set files are updated when a user manually selects a policy. An entry consisting of the context and the policy selected is added for every non-default policy chosen by the user. Currently, the context contains attributes that seem to have the greatest influence on
user preference: the task, the location, and the time of day. Additional context information can easily be added to the data set files.
Chapter 6

Evaluation

To evaluate the system, we used CAFE to create aggregations for two representative high-level tasks: playing an MPEG-formatted movie and editing a powerpoint presentation. We chose these tasks because they are common and because they can exploit appliance aggregation to provide a richer experience for the user. We used prototype representatives of the devices to create virtual ensembles, and we used CAFE to examine the effectiveness of using declarative policies to capture user preferences for aggregation. Here, we explain how the user interacts with the system to accomplish a task, and then we present some results about system performance.

Three types of devices are required to edit a presentation and play a video: a display device, an audio device, and an input device for the presentation. We used common sense to identify the important attributes of these devices and to assign values to the attributes. Further, we defined a set of declarative policies for the three policy levels: device, aggregation, and ensemble. Figure 6.1 shows the user interface for a task. A user can use the “Set Preference” option to bypass policy selection and directly specify an aggregation.

A user can manually select the policies at the different levels with the screen shown in Figure 6.2. CAFE shows only those policies that are relevant to the user’s requested task. A user can also specify a particular device. For example, the user may specifically want to use the projector as the display.

If the user does not want to specify the policies, the policy suggestion engine will select policies for the user. These policies are then used to rank the candidate aggregations as shown in Figure 6.3. If the user does not want to select an aggregation, CAFE will instantiate the best aggregation. Thus, by using the “Set Preference” option, a user can
enjoy one-click aggregation.

Our own experience with CAFE indicates that its choice of aggregations matches our intuition. In the relatively simple case of selecting an aggregation for a single request, CAFE's choice matched our choice. For two concurrent requests, CAFE's choice of devices for each aggregation consistently seemed optimal from our point of view. However, the fact that we developed the system and produced the declarative policies gave us in-depth knowledge about the policy weights and made us biased users.

Occasionally, the system would recommend unexpected policies. Close examination indicated that some piece of the context, such as the time of day, can become the critical splitting factor in the decision tree algorithm and cause an unexpected recommendation. This can be prevented by augmenting the history data to prioritize the weight of the various context factors in the recommendation process.

To evaluate the functionality of the system more rigorously and objectively, user studies will be quite useful. Such studies will involve users aggregating common devices to perform common task requests, such as the ones described above. We expect that the users
of the system will need to gain some familiarity with the effect of choosing policies on the outcome of aggregation. Further, some tuning will be required in the weights of the policies to adjust to individual preferences. A sound evaluation of the system and weights will include comparing the automatic choice of the aggregation by the system with a manual aggregation choice by the user. Conducting user studies was beyond the scope of our work so far.

6.1 Results

These experimental results demonstrate the efficiency with which CAFE is able to find and select an aggregation. Instantiation time is not included. All of these experiments were
Figure 6.3: Aggregation using policy suggestion engine.

performed using an iPAQ 3635 running Linux as the coordinator device.

Figure 6.4 shows CAFE’s response time for different size ensembles for one user task. The time taken for finding all aggregations shows the efficiency of running the JESS engine. The candidate selection time represents the cost of applying user-preferred policies to rank the possible aggregations. The graph shows that even for ensembles where the number of possible aggregations is quite large, the system is able to find all aggregations and rank them within a few seconds.

When a participating device from a running aggregation becomes unavailable, the cost of re-aggregation is shown by the device disappearance curve. It includes running the JESS engine on all of the available devices again, and ranking the possible aggregations. The re-aggregation cost is the sum of the time involved in the two steps mentioned above (finding
Figure 6.4: CAFE’s response time for handling one task. First two lines, ‘Finding all aggregations’ and ‘Candidate Selection’ show the corresponding response times for handling a new task. ‘Device disappearance’ line shows the response time for re-aggregation in case of disappearance of a participating device from a running aggregation.

all aggregations and candidate selection), but now the ensemble has one less device and CAFE has to apply a change penalty to account for user distraction. We found similar results for re-aggregation when a new device joins the ensemble.

For two tasks, the number of possible aggregations is even larger. Using the greedy heuristic described in Section 4.2.5, the system was able to determine the aggregations within ten seconds when the number of possibilities was more than a hundred, as shown in Figure 6.5. Because all of the steps involved are compute intensive, and because the resource-constrained iPAQ 3625 is acting as the coordinator, the observed response time is encouraging.

6.2 Lessons Learned

The following important lessons were learned through this work.

- The stability of an aggregation and the extent to which re-aggregation distracts a user are important concepts to capture and quantify. Quantifying these concepts and providing a means for comparing them with the quality of an aggregation has proven
Figure 6.5: CAFE’s response time for finding aggregations for two simultaneous tasks.

to be challenging.

- Descriptive, hierarchically structured policies are an effective way to elicit user preferences. This approach requires little input from the user, hides the numerical weights and other algebra, and only exposes user-oriented descriptions to the user.

- Simple descriptions are sufficient to capture device functionality for the purpose of automatic aggregation. The richer semantics provided by standards for service description, such as WSDL [1] and DAML-S [5], are not required here because we wanted to have a lightweight system that accommodates the resource constraints of mobile devices.

- Focusing on services at a high level has made it possible to avoid the complications of low-level constraints, such as protocol and format compatibility, wiring, and so forth. This has made it possible to focus on the more essential aspects of the problem. Our system can be easily extended to account for these low-level constraints by adding richer JESS rules.
• Using JESS to express device functionality and composition logic is an effective way to keep data and code together. JESS can be used to handle more complex compositional logic.
Chapter 7

Related Work

Many projects have tried to provide a richer experience to the user in the presence of multiple consumer appliances [13, 18, 19]. The basic idea that has been used is to represent devices as services, and then apply the techniques of service composition [16, 17, 19]. These projects have focused on resource requirements and conflicts, and have not accounted for user preferences and experience. By changing the focus from the resources to the user, our system provides a more personalized experience to the user.

The Metaglue project at the MIT AI Lab provides infrastructure for a multi-agent system in a smart-room environment. Describing a device in terms of the services it handles and needs in our system is similar to the way Metaglue describes the devices. The Rascal system, which is built on Metaglue, performs resource arbitration between multiple requests from multiple users in the context of an intelligent room [12]. There are two principal differences between Rascal and our approach that stem from the assumptions made about the system. First, in Rascal, requests come from multiple users, which makes it difficult to calibrate the requests against each other. In CAFE, the same user requests both tasks, which makes it possible for the user to employ a simple mechanism to assign preference to one of the tasks. Second, the issue of constraint satisfaction is critical in Rascal, which handles constraints on physical resources such as wires and switches. One of the shortcomings of Rascal is the time the system takes to satisfy additional requests. In CAFE, we assume a computing model that allows wireless connectivity between any two devices, and have effectively used our greedy heuristic to reduce selection time.

MAUT has been used by many projects [6, 9, 15] to elicit user preferences in different contexts. Iona [6] applies MAUT for mass customization of products. It realizes that user
may not be willing to answer all queries required to customize to product completely, so Iona goes for a best-effort approach. Iona asks queries in the order of their importance as long as the user is willing to answer, and then it applies linear-programming to best customize the product. [9] applies MAUT for news classification on the internet. It determines the utility of a news article by asking the user to select some categories from an ontology and to assign relevance values to them. A news item is rated according to its relevance with respect to categories, keywords and other context-specific attributes. CAFE differs from these systems in the way MAUT is applied. Since CAFE needs to compare the aggregation of devices in a dynamic environment, CAFE applies MAUT in hierarchical fashion (i.e. the three levels of policies). Also, CAFE introduces device stability and user distraction in the MAUT model, while other systems only focus on tangible device attributes to capture the user preferences.

Using context awareness to give personalized experience to the user in our system is related to the work by the Future Computing Environment group at Georgia Tech [4]. They use context in a touring application to predict what the user is observing and to provide information about that entity. In our system, we capture and apply context quite differently. We use it to predict a user’s preferences in an ad-hoc ensemble environment, based on the user’s past interactions with the system.
Chapter 8

Conclusion

This thesis presented the CAFE system. It provides an infrastructure to automatically aggregate device functionality in a user-centric way. We first identified the design requirements for a system to aggregate functionality in an ensemble. We then argued that the system should account for aggregation stability and user distraction in addition to aggregation quality. To capture user preferences, distraction metrics, and device properties in a generic way, we developed a hierarchical policy approach that uses three levels: device, aggregation, and ensemble. We found that the policies provide users with a higher level of abstraction, and that they help users manage numerous aggregation choices. We also determined that a policy recommendation engine minimizes the need for user intervention while performing aggregation, and allows a user to have a familiar experience even in an unfamiliar environment.

In the near future, we would like to learn how to capture user preferences and context more accurately. Currently, the relative weights used in a scoring policy definition are static, are assigned when the policy is defined, and can only be changed manually. The system should be able to adapt these weights automatically according to user preferences. Also, we would like to study the effect that adding new device attributes or advanced devices would have on the existing device scoring mechanism.
Bibliography


Engineering Track), Honolulu, Hawaii, May 2002.

