User-Centric Appliance Aggregation

Rajnish Kumar[1], Vahe Poladian[2], Ira Greenberg
Alan Messer, Dejan Milojicic
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-277
October 2nd, 2002*

aggregation, personalized, preference, context, history

As intelligent devices become affordable and as wireless infrastructure becomes pervasive, the potential to combine, or aggregate, device functionality to provide users with a better experience grows. However, even a small number of devices can be aggregated in many ways to perform a task. Currently, the user must choose among these aggregations without understanding essential information such as the properties of the devices. The problem is more severe in environments with devices that are unfamiliar to the user. A system for combining the functionality of devices in an ad-hoc environment to achieve a users desired experience is not available.

We present the design and implementation of a system for the user-centric aggregation of appliance functionality in an ad-hoc environment. This work supports the automated aggregation of functionality using predefined descriptions of devices, facilitates the selection of the best aggregation using declarative policies, and allows the 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 when using devices without having to worry about their details.

* Internal Accession Date Only
Approved for External Publication
[1] Rice University, Houston, TX
[2] Carnegie Mellon University, Pittsburgh, PA
© Copyright Hewlett-Packard Company 2002

User-Centric Appliance Aggregation

Rajnish Kumar†, Vahe Poladian‡, Ira Greenberg*, Alan Messer*, and Dejan Milojicic*

† Rice University, Houston, TX,
‡ Carnegie Mellon University, Pittsburgh, PA,
* HP Labs, Palo Alto, CA.

Abstract

As intelligent devices become affordable and as wireless infrastructure becomes pervasive, the potential to combine, or aggregate, device functionality to provide users with a better experience grows. However, even a small number of devices can be aggregated in many ways to perform a task. Currently, the user must choose among these aggregations without understanding essential information such as the properties of the devices. The problem is more severe in environments with devices that are unfamiliar to the user. A system for combining the functionality of devices in an ad-hoc environment to achieve a users desired experience is not available.

We present the design and implementation of a system for the user-centric aggregation of appliance functionality in an ad-hoc environment. This work supports the automated aggregation of functionality using predefined descriptions of devices, facilitates the selection of the best aggregation using declarative policies, and allows the 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 when using devices without having to worry about their details.

1 Introduction

Mobile consumer electronic devices are becoming ubiquitous. It is common for a user to own multiple computing devices, such as a laptop, a PDA, a digital camera, and a smart phone. In addition, a user often has access to fixed computing devices such as a desktop computer, speakers, a room projector, and a keyboard. These devices are 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 users experience.

Imagine that Alice, a traveling consultant, visits one of her clients on a business trip. She has access to a conference room during her visit that is equipped with a wall projector, a flat-screen monitor, a surround-sound system, and a set of tabletop subwoofer speakers. She has a personal laptop with built-in speakers, a PDA, and personal earphones. Alice wants to watch a company video using an application from her laptop (e.g., NetMeeting). 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, and she would prefer to listen to the sound privately, but without compromising its quality.

The available devices allow sixteen possible aggregations that satisfy her requirements. The number of possible aggregations would increase combinatorially if more devices joined her ensemble or if she tried to perform an additional task, as shown in Figure 1. To select the best aggregation, she has to know the properties of the devices, such as the size and color depth of the displays and the quality of the sound. It is not obvious whether the projector is the best choice for displaying the clip. Although it has the largest available screen, its color depth is somewhat limited. Choosing the correct sound device presents similar challenges. Alice had not even noticed the speakers in the conference room. While the earphones provide the best privacy, their sound quality is only optimized for stereo music. The laptops built-in speaker is optimized for playing digital streamed media, but it offers less privacy. Finally, the desktop speakers and the surround-sound system, although much better in terms of quality, are potentially too loud.

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. Further, she selects high-level policies, such as policies that favor user-oriented factors like privacy and a good display, and the system determines the aggregation that best matches Alices preferences. Interestingly, the system suggests that Alice should use the projector as the display even though it doesnt have the best quality because the system has discovered that the laptops

battery is low and will not last for the complete video. Alice accepts the systems suggestion, and feels 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 has to access her email and communicate with her friend. She uses her PDA to handle these tasks. The aggregation system, now running on the PDA, discovers the closest devices that are available to be aggregated in the way that Alice likes. The system knows Alices preferences by looking at her preference and execution history. 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 that can be employed by users in their everyday tasks. First, appliance functionality can be automatically aggregated to provide a user with a richer experience. Second, an intuitive mechanism helps a user select the best aggregation. The obvious aggregation may not be best for the user, and it could be complex for a user to determine the best aggregation even with a small number of devices because the number of possible aggregations can be large. Combined together, these features allow the user to have a familiar experience even in an unfamiliar environment with a minimal amount of interaction.

This report presents CAFE (Composition of Appliance Functionality in an Ensemble), a system that realizes the above scenario by employing a user-centric approach to automatically aggregate the devices available to a user. Using CAFE, a user can interact with the system declaratively at a high level by presenting preferences for the quality of aggregation that is desired. User preferences are captured as a set of predefined policies, which are then used to select an aggregation from several candidates, resolve resource conflicts that can arise when a user makes multiple requests, and adapt an aggregation if the environment changes.

CAF has the following highlights.

- Policies are used to capture user preferences about devices and aggregations as high-level abstractions. This allows the user to focus on the desired experience rather than understand the details of the devices and the system.
- The metric of user distraction is used to compare candidate aggregations when performing reconfigurations. Mechanisms for specifying and quantifying distraction have been identified. This information allows the user to specify trade-offs between the amount of distraction the user is willing to tolerate and the quality of aggregation the user desires.
- Policies can be suggested to the user based on context and the users aggregation history. This allows aggregation to be similar and predictive across different environments, and requires even less input from the user.
- Optimizations are used to reduce the cost of selecting aggregations when a user simultaneously requests multiple tasks.

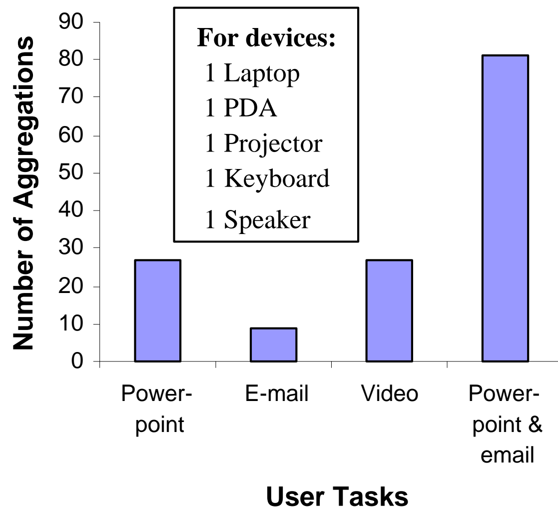

Figure 1: *The number of possible aggregations increases combinatorially with an increase in the number of devices or tasks. The number of aggregations were computed by applying simple composition rules over the devices listed in the diagram.*

This report is organized as follows. Section 2 and 3 present the design goals and the architecture of the system. Section 4 describes the implementation. Results, in terms of evaluation of the system and lessons learned are included in Section 5. Related work is described in Section 6. In Section 7, we summarize the report and suggest possible future work.

2 CAFE System Model

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

2.1 Definitions

The following terms are used throughout the report.

Ensemble: A set of appliances that can be accessed and controlled by the user. These appliances are either owned

by the user or borrowed temporarily, and can be considered available for the purpose of performing a users tasks.
Aggregation: A subset of the appliances in an ensemble that are used together to 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.
Distraction: The inconvenience caused to a user by changing an aggregation. 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 users current task.
Stability: A metric that quantifies how well an aggregation will be able to perform a task to completion. For example, if a devices battery will be exhausted before a certain task is completed, then any aggregation that uses that device will have lower stability than an aggregation that uses another device with a longer battery life. Similarly, an aggregation will have relatively low stability if contains a borrowed device that is likely to become unavailable before the task is completed.
Context: Any information that can be used to characterize an ensembles environment citeDey2001. Examples include the user task, the ensemble devices, the users location, and the time of day.
Policy: A group of numerical weights and a high-level description that encode a users 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.

2.2 System Assumptions

Device and Service Model

A device provides 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 devices availability and functionality, and for providing information about the devices dynamic properties. This representative can execute on the device itself, or on some other device.

Application Service Model

CAFE is designed to handle user requests that can profit from the aggregation of multiple devices in an ensemble. Its user-centric focus will be useful when there are many possible aggregations for a particular user request. We assume that applications are component-based and that they expose interfaces for remote activation and easy integration with other components or devices. Application components are remotely activated by the system to initialize an aggregation. The interfaces should clearly define the signature of the activation methods, and the interface definitions should be specified in a device-independent language.

An applications 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

To make the decoupling between an applications control logic and its input and output devices more realistic, we assume that all devices are connected wirelessly. We believe that the set of devices that comprise an ensemble can change relatively frequently. For example, devices can be turned on and off, can be borrowed and returned, can run out of power, can lose network connectivity, and can break.

Infrastructure Support

We assume the existence of middleware that allows applications to be remotely started on devices, similar to what is provided by systems like Metaglue [8]. CAFE will determine the aggregation that is closest to a users 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 while the task is running.

2.3 Design Goals

CAFE provides support for the automatic aggregation of devices and for the user-centric selection of an optimal aggregation. Here, we discuss the design goals for these objectives.

Goal: Allow automatic aggregation of devices, and among all possible aggregations, allow selection of the one that best matches user preference.
Design: To allow automatic aggregation, we view each kind of functionality provided by a device as a service and we describe devices as the services it can handle and the services it will need. We represent a service as a data-action pair. This approach is similar in approach to MIME types used in Internet messaging. For example, 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: We achieve this goal by employing stability and distraction as metrics in the selection of the best aggregation. When selecting an aggregation, we quantify the stability of candidate devices and allow the user 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 because a user would like to have a stable working environment for the duration of a task. Thus, it is important to be able to consider stability and balance it against aggregation quality. Sometimes the user will consider the gain in quality to be large enough to outweigh the loss of stability.

When we are forced to re-aggregate, we quantify and penalize changes that are potentially disruptive to the user, and allow the user to specify trade-offs between aggregation quality and distraction. We believe that 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, we must re-aggregate, and do so in such a way that distraction is minimized. 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. Users are generally willing to tolerate delays of up to 10 seconds for responses to interactive queries. Ensuring this response time presents challenges on resource-contrained platforms such as PDAs. From personal communication with members of the Rascal project, we anticipated that handling multiple simultaneous requests in an acceptable amount of time is especially desirable. We propose using optimized heuristics to handle the case of two concurrent requests.

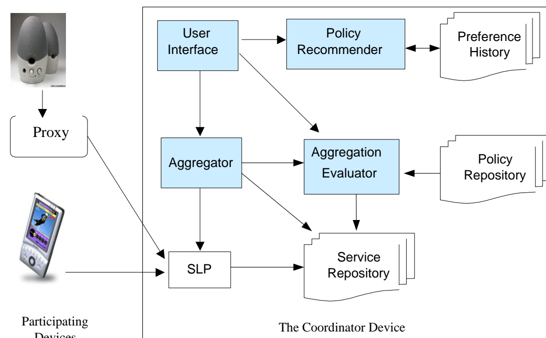

Figure 2: *The CAFE Architecture.*

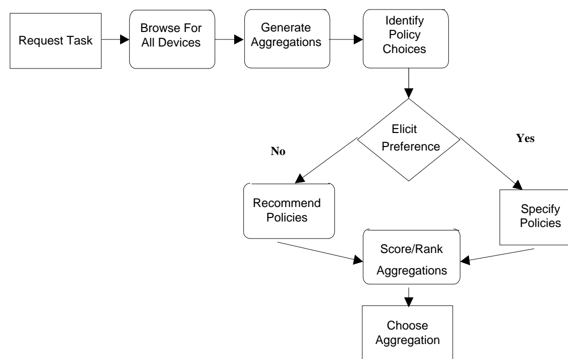

Figure 3: *Steps involved in finding the best aggregation.*

3 CAFE System Architecture

CAFE is designed to provide an infrastructure for appliance functionality aggregation in a user-centric way. In this section, we present the CAFE architecture and explain the mechanisms used to calculate candidate aggregations and select the optimal one.

The CAFE architecture is shown in Figure 2. The entire system infrastructure runs on a specifically selected device that we call the Coordinator. Any member of the ensemble can act as the Coordinator and is responsible for running the five components shown in Figure 2. Shaded (blue) boxes represent run-time components that we have developed. The components inside the dotted box represent the entire system. One of the devices, for example user's PDA or laptop, is selected to run this system. We call this device the Coordinator. Following are the main components of CAFE system :

1. **SLP Provider:** registers and discovers services
2. **User Interface:** provides a Web-based user interface
3. **Aggregator:** calculates all possible aggregation candidates for a given user task

4. **Candidate Selector:** selects the best aggregation among the candidates based on user preferences

5. **Policy Suggestion Engine:** predicts the user's preferences based on user preference and execution history

3.1 Major Activities

Figure 3 presents the high-level flow of the system including interactions with the user and major computation steps. We now discuss these major activities.

User request presentation

We have developed a simple, Web-based interface for user request entry. A user requests a task by specifying a data-action pair. For example, to request that the system play the movie "The Matrix", the user would specify the directive "play" and the data source "TheMatrix.mpeg". Further, the user has the option to guide the system in the selection of the optimal aggregation. But the user does not have to specify preferences, as the system is capable of predicting user's preferences based on history.

Device description and registration

A Service Lookup Protocol (SLP) is used for device registration. Currently, devices register their functionality with the Coordinator application by posting their service description file and length of availability. The coordinator then makes the registration to SLP registration on the behalf of the devices. Implementation choice was made to allow the system to be notified of new registrations and (de-registrations). We use a simple XML format for describing the services offered by the device, various attributes of the device, and parameters required for execution.

Devices are described in terms of the functionality they support, in particular, by a data-action pair. For example, a device that can play mpeg movies is described as being able to handle the "play, mpeg" pair. Further, for each directive supported, devices also describe their needs. A device supporting a "play, mpeg" directive may need a sound device and a video device to function. Using this information, the system 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 runtime invocation information, such as the handle of the executable that needs to be run.

Automatic Aggregation Generation

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 the system using the existing device descriptions. The data-action pair in the user's request is taken as an unsatisfied need, and expanded. The expansion of an aggregation includes adding a device that is able to handle an unsatisfied need in the current state of the aggregation. As new devices are added to the aggregation, some needs become satisfied, but new needs may also be added. Aggregation expansion stops when all the needs are satisfied. At this point aggregation is complete, or final. If at some point we run out of devices, and the aggregation is not final, then the request cannot be satisfied.

The core of the Aggregator module is Jave Expert Shell System (JESS), a rule-based engine written entirely in Java [6]. Device functionality is expressed as facts, and rules are used to allow the aggregation of compatible devices. As mentioned earlier, device functionality descriptions contain needs, which are used to generate aggregations. A user's request is also described as a fact that needs to be satisfied. Asserting the user's requests starts the chaining of rules, which ultimately completes with the generation of aggregations that completely satisfy the user request.

Candidate Selection

Once the Aggregator generates all candidate aggregations, the Selector module is invoked to rank the aggregations according to the preferences of the user. User preference is captured by high-level, declarative policies that describe the device properties abstractly. Section 4.3 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 to allow one to be selected, or the system can automatically instantiate the best candidate.

3.2 Scoring Mechanism

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

Figure 4 gives a high level picture of the scoring mechanism. Devices are scored according to the values of their attributes and user's preference. Aggregations are scored by combining the scores of the participating devices and

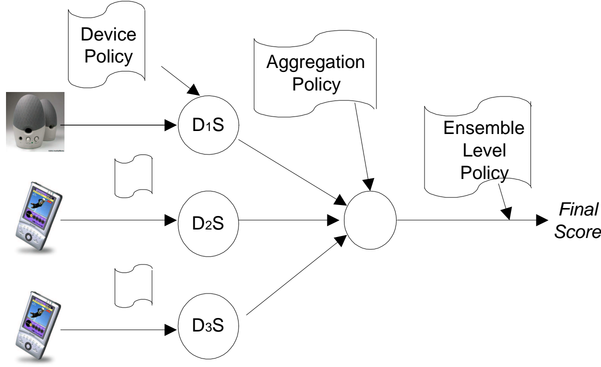

Figure 4: *Scoring hierarchy. The final score of the aggregation is the weighted sum of the participating devices' scores according to the aggregation policy. If this aggregation is the result of re-aggregation because of some change, the final score is adjusted to account for user distraction. This adjustment is applied by using an ensemble-level policy.*

user's preferences. And ensemble score is computed by combining the scores of the aggregations and less tangible metric such as inconvenience to the user. In the next few sections we describe the details of the scoring.

Device Policies and Device Scoring

Device-level policies allow the scoring of the devices and comparing of two devices, which provide the same type of service. Examples of devices offering the same type of service are: a room speaker, a desktop speaker, an earphone. For each type of device, we have identified a set of properties that are used for scoring. As an example, for sound devices, we have selected the quality of sound, whether or not sound can be heard in privacy, whether or not speaker has sub-woofers, and whether or not sound is surround.

We have identified a number of discrete values for each attribute, at a coarse level of granularity. For example, for sound quality we have chosen five values ranging from "low" to "very high". For each value, we assign a score between zero and one hundred. We 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.

Notice that these scores are system-wide. This means that the same set of scores is applied regardless of time, place, context. This approach avoids asking for user input for every value of each attribute, which may be tedious and take a long time. Further, because users do not generally think in terms of scoring different values, eliciting scores for every possible value of each attribute may not yield meaningful results. Instead, we obtain a user's preferences by asking the user to select a policy from a pre-defined group.

A device scoring policy captures user preference. We have created several sample policies. In addition, the system allows creation of new policies. This can be done by developers and users.

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 that the device gets for its attributes. A device score is computed as:

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

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 5 gives an example of applying device scoring policies to compare the sound devices. Notice that depending on the choice of policy, the best device varies. When the quality of sound is preferred, the room speaker is best; when privacy is most important, the earphones are best; when both privacy and quality are equally preferred, the desktop speaker is the best.

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 deal with numbers. Several aggregation-level policies are provided with the system for various common tasks.

An aggregation score is computed as:

$$AS(A, AP) = \sum_{i=1}^{n} sw_i(D, AP) * e(D_i) * DS_i(D, DP_i)(2)$$

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

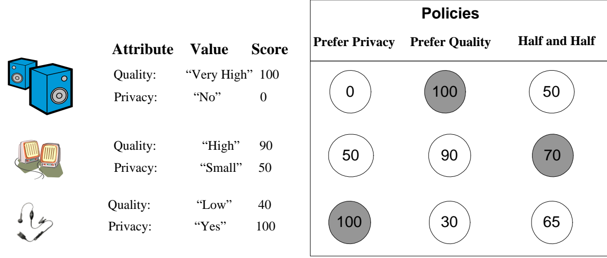

Figure 5: *Example for scoring sound devices using a service scoring policy for playing sound when quality and privacy are the issues being considered. The numbers in the ovals give the score for the device after applying the service policy.*

device D_i. We will discuss this metric further in the next section.

Stability and Future Planning

Once an aggregation is instantiated, changing it will 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 some 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, we believe that such a metric can be computed in several ways. One approach is to collect history of 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, the advertised length of the conference, or by asking the user, and then compute availability accordingly.

In formula (2), $e(D_i)$ 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 probability of that device becoming unavailable.

To account for possible user inconvenience when there is a change in devices that are participating in a running aggregation, we weigh each device's score based on stability. Stability is the probability that an aggregation will be able to perform a task for the desired amount of time. We multiply the quality score of the devices by a percentage indicating its stability, and use this product in scoring the device in an aggregation.

Adaptation to Changes

Although CAFE is future planning and it 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 unavailable, 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 candidates, we use penalties 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 connection 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 requires re-aggregation to be considered. Also, note that re-aggregation may not actually occur, e.g., if the distraction penalty exceeds the increase in quality.

We believe that minimizing a user's inconvenience is an important factor to consider while adapting an aggregation. For example, a user may want to avoid having the display change, or a user may want to avoid moving to use a different microphone. On the other hand, moving the MPEG decoder from one computer to another may not bother the user much. We believe that changes to some devices may cause more inconvenience to the user than changes to 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.

In order to compute user's inconvenience, we compute a penalty score, which quantifies the amount of inconvenience that the user has to incur. We call this *aggregation difference penalty*. The formula for computing that measure is:

$$ADS(A, A) = \sum_{i=1}^{d} b_i(S_i, S_i) * DDP_i (3)$$

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 inconvenience that the user has to incur, should the change in devices occur. Notice that there are situations, in which the user may like to minimize the amount of change at the expense of quality. In other situations, user may 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 devices that are included in 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 raising the possibility of a change.

A user's tolerance for change is represented by weights for distraction penalties and aggregation quality.

$$ES_R(A) = AS(A, AP) - q_d * ADS(A, A), \quad (4)$$

where AS is the aggregation score for A according to aggregation policy AP, q_d is a weight between 0 and 1 given to the inconvenience factor, and $ADS(A, A)$ is the difference score between 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 difference, q_d is 0.5.

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, the user needs to specify the importance of the tasks relative to each other. For now, we consider only two tasks, and there are five choices: "Favor first task strongly", "Favor first task", "Equal", "Favor second task", and "Favor second task strongly". These choices represent different weights for the tasks. For strong preference the ratio is 0.9 to 0.1, for weak preference the ratio is 0.7 to 0.3, and for equal preference the ratio is 0.5 to 0.5. Note that the weights always add to one.

We compute an ensemble-wide score as follows:

$$ES_R(EP, A_1, A_2) = q_1 * AS(A_1) + (1 - q_1) * AS(A_1), \quad (5)$$

where EP is the ensemble's favoring policy, $AS(A_1)$ 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.

We have two approaches for generating optimal 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, unless the user is willing to settle for a sub-optimal choice, ensuring that the best aggregation has been found requires complete enumeration of all candidates.

In the second approach, that we call 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.

Notice that the formulae involved in calculating the device, aggregation, and ensemble scores are linear. Using this observation, we can compute the potential contribution of a given device to the final ensemble-wide score, if that device is used to satisfy a particular request (e.g. first request, second request, etc). Consequently, we can arbitrate device of the same type among different requests by computing each devices contribution, and then by calculating the most efficient allocation. This calculation takes considerably less time, then the naive approach.

Policy Recommendation Engine

Policy recommendation engine provides support for predicting user's preferred policies for devices, aggregation, and ensemble levels. This allows CAFE to aggregate devices even when the user chooses not to 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 (history), 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. User past preferences and contexts are stored in history files, and these files are used as the learning set by the decision tree algorithms. Context contains the factors that may influence user's choice of policies, e.g. user task, or location. The history files are updated whenever the user provides her preferences manually. Hence, in the beginning the history files may not be rich enough to help the decision tree algorithm to provide consistent prediction. When the user's history is not rich enough to make a meaningful decision, default policies that are prespecified for each of the three policy levels are used. More about this will be discussed in implementation section.

4 Implementation

In this section, we will describe the implementation of the components of the system and the experience learned from the implementation.

The system is implemented in Java and runs within a Java servlet engine. The interface of the system is Web-based and can be used from any device within the ensemble. The system component itself is centralized. It is discoverable, which makes the system more flexible in ad-hoc environments.

System Components and Initialization

The bulk of the system is composed of Java code, XML

configuration files, and a JESS template file. Upon initialization, the system loads the JESS file and initializes the JESS engine. It then loads the XML configuration files, and prepares to accept requests.

The JESS template file contains definitions for fact templates and several rules for expanding aggregations. There are templates for the following objects: requests, services, aggregations, and final-aggregations. More details on these templates are forthcoming.

All of the XML files are kept in a repository. These files include service descriptions, policies, and history.

Service Description and Registration

A device registers by POST-ing service description XML files to the registration URL. A service description XML file specifies the data type and action directive pair that the service can handle, the other services that the service requires, and additional information such as values for attributes that are appropriate for that service type. Here is a sample service description file.

```
<service name="MPEG Handler Splitter Application - Splits Sound
and Video Into Separate Streams" uniqueId="MPEGPlayerSplitter">
    <handles mime="mpeg" action="play"/>
    <virtualLocation>polian.hpl.hp.com</virtualLocation>
    <executableHandle>splitter.bat</executableHandle>
    <serviceReqParameters>mp3player=$serviceReq:mp3:
        input</serviceReqParameters>
    <serviceReqParameters>aviPlayer=$serviceReq:avi:
        input</serviceReqParameters>
    <requires>
        <serviceReq mime="mp3" action="play"/>
        <serviceReq mime="avi" action="play"/>
    </requires>
</service>
```

The system runs an XSLT [3] transformation on a service description file to generate registration information for SLP. (We use OpenSLP, a freeware implementation of SLP.) Because the formats involved are simple, it would be possible to develop custom format translators that do not use XSLT. This would be beneficial because the XSLT libraries are rather 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.

The system uses the registration string to register the service with the SLP system. Service registration remains valid for a fixed duration, the expiry interval, after which the service registration expires. Devices are expected to re-register themselves with SLP directory server within this expiry interval. This is enforced to ensure that the information in the system is up to date. The length of the expiry interval is configurable and depends upon how ad hoc is the environment. The disadvantage of enforcing expiry interval is that the devices are forced to periodically re-register.

Task Request and Satisfaction

A user requests a task by specifying a data file and action directive pair. The extension of the file is used as the mime type of the request. When the request is received, the system 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. SLP provides a function for querying all of the registered service types. It also provides a function for querying all of the services for a given service type. This allows all of the registered services to be browsed without any prior information. We initially used this approach, but later we changed the implementation to cache this information in memory, using the java.util.Hashtable object. Browsing the SLP directory can be slow because the API makes a multicast network request, and the multicast requests 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_avi ))
))
```

This fact is an instantiation of the following JESS template definition for a service object.

```
( assert
(deftemplate service
    (slot serviceId)
```

```
        (slot handlesDirective)
        (slot final (default yes))
        (multislot requires (default (create$)) )
)
```

A template definition basically defines the name of the type and the slots that the type has. Note that the "requires" slot is a multislot, which means that multiple values can be specified there. For the service we have presented, there are two requirements: "play_mp3" and "play_avi".

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. Here is the template definition for an aggregation.

```
(deftemplate aggregation
        (slot handlesDirective)
        (multislot requires (default (create$)) )
        (multislot requiredBy (default (create$)) )
        (multislot members (default (create$)) )
        (multislot finalMembers (default (create$)) )
        (multislot pinnedDevices (default (create$)))
        (multislot pinnedDevicesHandle
(default (create$)))
)
```

We currently do not consider using more elaborate matching rules that can allow protocol and format compatibility to be checked when chaining services together. Because JESS has powerful pattern matching support, our current implementation can be extended to account for elaborate matching while chaining the services together. Here is an example of an aggregation expansion rule.

```
(defrule expandsNormalServiceNotPinned
?aService < − ( service (serviceId ?theId)
(handlesDirective ?firstReq)
(final no)
(requires $?theNewReqs))
        ?anAggregation < − ( aggregation
                (handlesDirective ?theDirective)
                (requires ?firstReq $?restReqs )
                (members $?mems)
                (requiredBy $?ReqsSvc )
                (finalMembers $?finalMems))
(test (isNotInList ?theId $?mems))
(test (isNotInList ?firstReq $?pHandles))
=>
( assert ( aggregation
        (handlesDirective ?theDirective)
(requires $?restReqs $?theNewReqs)
(members $?mems ?theId)
        (requiredBy $?ReqsSvc
(makeMultiSlot ?theId $?theNewReqs))
        (finalMembers $?finalMems)
        (pinnedDevices $?pDevices)
        (pinnedDevicesHandle $?pHandles)
)))
```

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 several other features, including the "pinning" of devices where a user is able to specify the exact devices to use to perform some of the services.

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.

After aggregations are generated, the system scores them using a hierarchy of policies. The user has two ways to choose policies. First, the user can let the system automatically choose the policies. The system makes its decisions by applying a decision-tree classifier to the users history and context. Second, the user can select specific policies. Any policies not specified by the user are filled in automatically. The policies are used to score the aggregations.

Aggregation Scoring

To score aggregations, the system needs to know 1) how to score each participating device, 2) how to assign weights to the devices in the aggregation, and 3) if system needs to apply ensemble wide policy to account for distraction in case of reaggregations.

For each type of service, attributes are identified that are common across all devices support that service type. For example, for devices that support the display-video service, possible attributes are the size of the display, the flatness of the screen, and whether the display allows private viewing.

For each identified attribute, common sense is used to score all of its possible values. For example, for the size attribute, five values have been identified: extra large, large, medium, small, and very small. Here is the service scorer XML file for display video service.

```
<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">
            <point value="yes" score="100"/>
            <point value="no" score="0"/>
      </attribute>
```

```
        <attribute name="wallMounted">
            <point value="yes" score="100"/>
            <point value="no" score="0"/>
        </attribute>
        <attribute name="private">
            <point value="yes" score="100"/>
            <point value="no" score="0"/>
        </attribute>
    </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 the privacy attribute. Here is a sample display video scoring policy.

```
<serviceScoringPolicy
name="Largest display, slight
preference for flat screen"
action="display" mime="video"
id="Policy_Larger_Better">
<attributes>
    <attribute name="size" weight=".9"/>
    <attribute name="flatScreen" weight=".1"/>
    <attribute name="wallMounted" weight=".0"/>
    <attribute name="private" weight=".0"/>
</attributes>
</serviceScoringPolicy>
```

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 system can have normalized scores for different devices in the ensemble. These policies allow users to weigh some attributes more than others and add flexibility to the system, without requiring them to provide a significant amount of input.

An aggregation scoring policy is 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). An ensemble-wide score is then computed for each aggregation using its aggregation quality score and stability metric. The stability metric is based on the average availability of the devices that are part of an aggregation. Currently, a history-based approach is used to gauge how frequently the devices have been available. The ensemble-wide availability of an aggregation is computed as the lowest availability value of any device in the aggregation.

An ensemble-wide policy is used to combine the aggregation quality and stability scores. This policy simply specifies which dimension should get more weight. By giving substantial weight to stability, quality is sacrificed for a more stable aggregation. The rational behind this decision is simple: if a device that is part of a running aggregation becomes unavailable, then it will be necessary to re-configure the system, a potentially costly operation.

Another measure in the ensemble-wide policy is the change penalty weight. This measure is only used during reconfiguration.

Policy Suggestion

We used the implementation of decision tree algorithm provided by WEKA [2] for the 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 users 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 ARFF format (attribute-relation file format), the format expected by WEKA. Here is an example of an ARFF file for predicting a device-level policy for the "display_video" task.

```
@relation policyEngine
@attribute task {play_mpeg, play_avi, 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, e.g., 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 we believe 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.

5 Evaluation

To evaluate the system, we used CAFE to support aggregations for two representative high-level tasks: playing an mpeg-formatted movie and editing a powerpoint presentation. We chose these tasks because they are commonly

Figure 6: *Request Input Page.*

performed and leverage appliance aggregation to provide richer experience for the user. We used decoy representative of the different devices to have virtual ensembles, and we used CAFE system to see the effectiveness of the declarative policies to capture user preferences for aggregation. Here we explain how the user interacts with the system to accomplish her task, and then we give some results about the system performance.

To accomplish the tasks of editing a presentation and playing a video, three types of devices are required, the display, audio and input device for the presentation. We used common sense to identify important attributes of these devices and to score the different values of these attributes. Further, we defined a set of declarative policies for different levels: device, aggregation and ensemble. Figure 6 is the user interface to input the task. User can use "Set Preference" option to bypass the policy selection phase and directly obtain the aggregations.

If the user decides to manually select the possible policies at different levels, she can do so by selecting them at the screen shown in Figure 7. CAFE shows only those policies for selection which are relevant to the user requested task. User can also pin a particular device, e.g. she may specifically want to use the projector for display.

If the user does not want to specify the policies, the policy suggestion engine predicts the user choice for policies, and these policies are used to rank the possible aggregations as shown in Figure 8. We here show the sorted list of possible aggregations for explaining the system, though the best aggregation can be directly instantiated without asking user to select one out of those possible sorted aggregations. Thus, by using "Set Preference" option, user

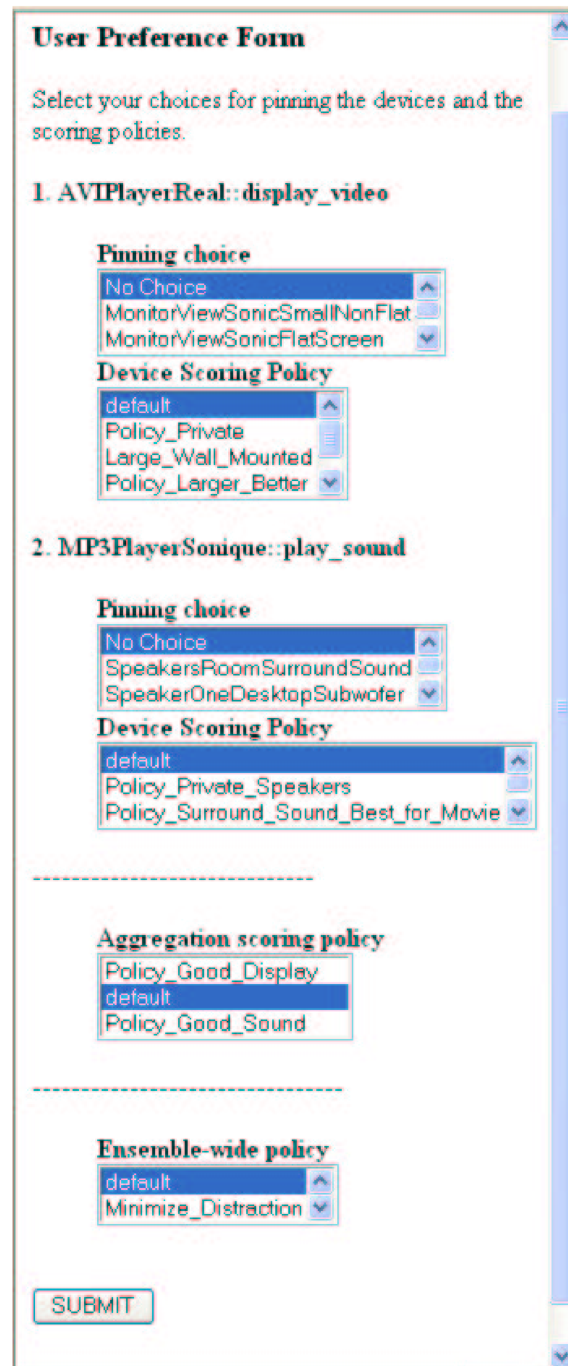

Figure 7: *User preferences form page.*

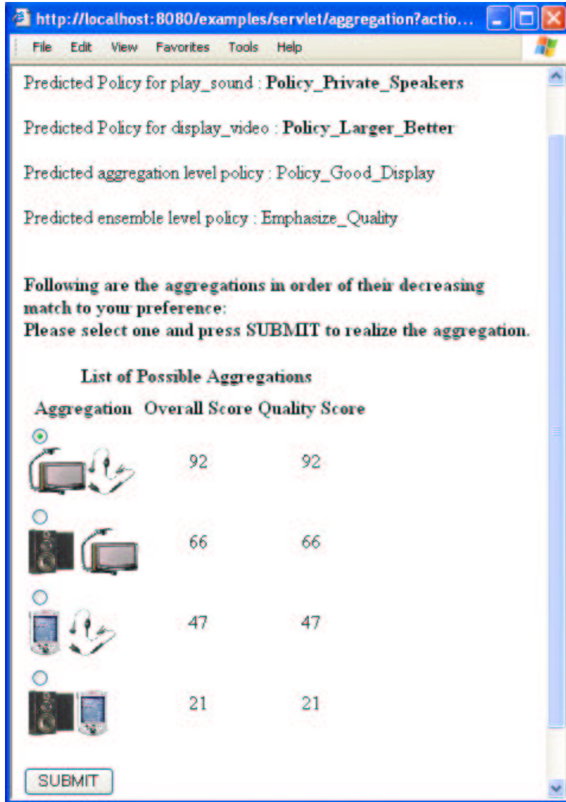

Figure 8: *Aggregation using policy suggestion engine.*

can enjoy one-click aggregation for her requests.

Our own experience with the system indicates, that CAFE's choice of aggregations matches our intuition. In a relatively simple case of one-request aggregation, the system's choice of best aggregation matches our desired choice. In case of two concurrent request, the systems choice of devices for each aggregation seemed consistently optimal from our point of view. However, we should mention that as developers of the system and producers of the declarative policies, we have in-depth knowledge of the weights of these policies. As such, we are somewhat biased users. In case of policy prediction, occasionally we would encounter somewhat unexpected recommendations from the system. Close examination of the reasons indicated that time of day, for example, can become the critical split in the decision tree algorithm, resulting in an unexpected recommendation. We propose further organizing the history data in such a way, so as to prioritize the weight of the various context factors in the recommendation process.

To evaluate the functionality of the system more rigorously and objectively, we believe that user studies might be required. Such studies will involve users wishing to aggregate for 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 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 project.

We would also like to address the rationale of choosing weighted sum for computing scores of devices, aggregations, and the ensemble. The Multiattribute Preference Model [9], which is a well-established model in modern microeconomic theory, suggests that utility across multiple objectives can be combined using weighted sums, provided that these objectives satisfy the independence assumption. Further, this model also provides mechanisms for eliciting utility for the various levels of each attribute, provided that these attributes are independent. We have essentially adopted this model for the purposes of scoring device attribute values, devices, aggregations, and the ensemble. We believe that independence assumption holds with respect to the functionality offered by different device types, since different device types offer different services, that are not substitutable for each other. It is harder to argue the independence of different attributes of one device. In particular, there maybe attributes, which are close substitutes. However, we would like to emphasize that the willingness of the user to substitute a significantly large size of the display with lower color depth is NOT sufficient to refute that size and color depth are independent. In general, the independence of attributes can be proven by having elicited a complete utility profile of the user, and then applying independence tests.

Results Below we present experimental results regarding the efficiency of CAFE. All of these experiments were performed using an iPAQ 3635 running Linux as the coordinator device. Since here we want to measure the efficiency of CAFE in finding and selecting the aggregation, these results do not include instantiation time.

Figure 9 shows CAFE's response time for different size ensembles for one user task. 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 new device joins the ensemble or an existing device leaves the ensemble, the time taken to do reaggregation is again within acceptable limits.

For two tasks, the number of possible aggregations is even larger. Using the greedy heuristic described in Section 3.3, 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 10.

Lessons Learned The following important lessons were

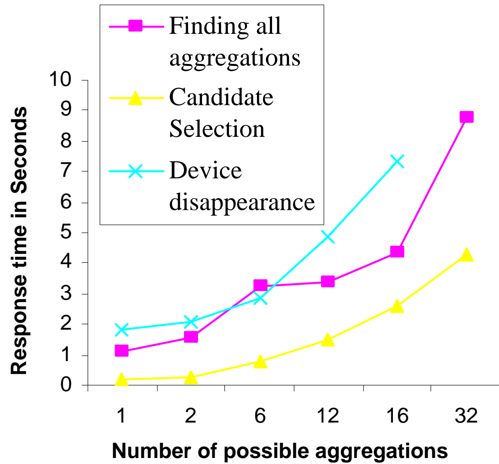

Figure 9: *CAFE's response time for handling one task. First two lines, 'Finding all aggregations' and 'Candidate Selection' shows the corresponding response times for handling a new task. 'Device disappearance' line shows the response time for reaggregation in case of disappearance of a participating device from a running aggregation.*

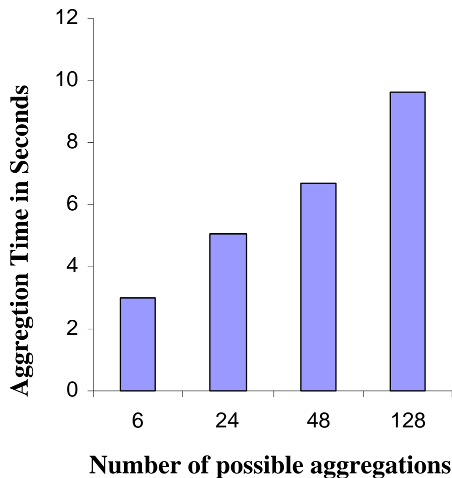

Figure 10: *CAFE's response time for finding aggregations for two consecutive tasks.*

learned through this work:

- The stability of an aggregation and the distraction of re-aggregation to a user are important concepts to capture and quantify. Quantifying these concepts and providing means for comparing them with the quality of an aggregation has proven 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 for our purpose because we wanted to have a light-weight system considering 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. We believe that JESS can be used to handle more complex compositional logic.

6 Related Work

Many projects have tried to provide a richer experience to the user in the presence of multiple consumer appliances [8], [12], [13]. The basic idea that been used is to represent devices as services, and then apply the techniques of service composition [13], [10], [11]. These projects have focused on resource requirements and conflicts, have not accounted for user preferences and experience. By changing the focus from 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 multi-agent system in a smart room environment. Describing a device in terms of *handle* and *need* in our system is similar to the way Metaglue describes a device as resources. The Rascal system, which is built on Metaglue, does resource arbitration between multiple requests in the context of an intelligent room [7].

There are two principal differences between Rascal and our approach that stem from the assumptions of the system. First, in Rascal, the requests come from multiple users, making it difficult to calibrate the requests against each other. In our case, the same user requests both tasks, making it possible for the user to employ a simple mechanism to give preference to one or the other task.

Second, the issue of constraint satisfaction is critical in Rascal. Rascal handles constraints on physical resources such as wires and switches. One of the shortcomings of Rascal, based on personal correspondence with the authors, is the time the system takes to satisfy additional requests. In our work, we have assumed a computing model that allows wireless connectivity between any two devices.

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 ensemble environment, based on the user's past interaction with the system.

7 Conclusion

In this report, we presented the CAFE system. It provides infrastructure for the automatic aggregation of device functionality in a user-centric way. We first identified the design requirements for such a system to do functionality aggregation in an ensemble. We then argued that such a system should account for user distraction and aggregation stability, 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 policies provide users with a higher level of abstraction and that they help users manage large numbers of aggregation choices. Using a policy recommendation engine minimizes the user intervention while performing aggregation, and also allows user to have similar experience even in different and new environments.

Some of the possible future directions in which our current work can be extened is discussed below.

More accurate capture of user preference and context

In the current approach to scoring policy definition, the relative weights of the device attributes are static and assigned at the beginning when the policy is defined. A user can tune these weights according to preferences, but it is unlikely that a user will want to tune all of the weights. Thus, the relative weights used in the policies are static and they do not reflect the difference in different user's preferences. That is, for some user the change from "large" display to "very large" display may not be as important as the change from "medium" to "large" display, while for another user the opposite may be true. The system should adapt these weights automatically according to user preferences.

Employing a feedback mechanism can help with the automatic tuning of scoring policy weights. One simple approach may be to ask a user about the aggregations when the user does not want to follow the CAFE's suggestions.

Applying to other application domains

The idea of using policy as an abstraction to elicit user preferences can be applied to domains other than appliance aggregation. One potential application of the policy framework is data utility centers. The data utility center provides data as a utility service, and it can encode the system properties as policies. For example, the properties can be data privacy, data access time, reliability, availability, and so forth. The system manager can design policies using these properties in different combinations, and a user can express how data should be stored by selecting these high-level policies. Based on user preference, the system can dynamically decide how to store and manager the user's data. We are exploring how to give relative weights to these properties and what are other important properties to consider for capturing user preference and needs.

Making device scoring more flexible

We are looking into the effects of adding new device attributes or advanced devices on the existing device scoring mechanism. We believe that the manufacturer of the new device can provide appropriate standard values for the attributes that we wish to evaluate. Furthermore, as technology advances and better devices become available, it should be possible to automatically calibrate the scoring to make the highest value equal to 100, and appropriately scale down all of the other values. While this approach is naive, it certainly provides a simple way of dealing with newly introduced devices.

References

[1] Web services description language(wsdl) 1.1: http://www.w3.org/tr/wsdl.

[2] Weka 3 machine learning software in java: http://www.cs.waikato.ac.nz/ ml/weka/.

[3] Xsl transformations(xslt) version 1.0 : http://www.w3.org/tr/xslt.

[4] G. D. Abowd, A. K. Dey, R. Orr, and J. A. Brotherton. Context-awareness in wearable and ubiquitous computing. In *Proceedings of the 1st Inter-*

national Symposium on Wearable Computers ISWC, pages 179–180, 1997.

[5] A. Ankolekar, M. Burstein, J. R. Hobbs, D. M. Ora Lassila, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. Challenges: an application model for pervasive computing. In *Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002)*, pages 266–274, Sardinia, Italia, 2002.

[6] E. J. Friedman-Hill. Jess, the java expert system shell. Technical Report SAND98-8206, Sandia National Laboratories, 1997.

[7] K. Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of The Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, Krakow, Poland, 2001.

[8] K. Gajos, L. Weisman, and H. Shrobe. Design principles for resource management systems for intelligent spaces. In *Proceedings of The Second International Workshop on Self-Adaptive Software*, Budapest, Hungary, 2001.

[9] R. L. Keeney, H. Raiffa, and R. Meyer. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Cambridge Univ Press, 1993.

[10] S. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR '02)*, Toulouse, France, 2002.

[11] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *Proceedings of The Eleventh World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, May 2002.

[12] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: A service framework for ubiquitous computing environments. *Lecture Notes in Computer Science*, 2201:56–??, 2001.

[13] S. C. Samuel. Ninja paths: An architecture for composing services over wide area networks.