

Selecting Devices for Aggregation

Rajnish Kumar¹, Vahe Poladian², Ira Greenberg³, Alan Messer⁴, and Dejan Milojicic⁵

¹ Georgia Institute of Technology, Atlanta, GA, ² Carnegie Mellon University, Pittsburgh, PA,

³ Consultant, Mountain View, CA, ⁴ Samsung Electronics CTO/SISA, San Jose, CA ⁵ HP Labs, Palo Alto, CA.

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. Often, there will be multiple devices providing similar functionality that the user will have to choose from for the aggregation. This paper presents the design and prototype implementation of a system for automatic selection of devices for aggregation in a dynamic environment. 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 the device and the environment details.

1 Introduction

Mobile consumer electronic devices are becoming ubiquitous, and users now own several of these devices, such as a smart phone, a digital camera, a portable music player, and a laptop. 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.

If the individual devices are combined together as an aggregation, they will offer much greater functionality and will significantly enhance a user's experience for a given task. For example, a projector, ear-buds, and a PDA can be combined to form an aggregation for the purpose of playing a movie in MPEG format. Recent research projects have looked into the different aspects of device aggregation [14, 6]. For example, AAA (*Appliance Aggregation Architecture*) [14] presents a user with a common model of ownership, shared state, shared applications, and shared functionality for a group of devices.

This paper aims to extend the aggregation infrastructure with the capability of automatic selection of devices for aggregation. We present below a motivating example

to show how the device selection for aggregation is non-trivial and how an infrastructure support for the automatic selection will be helpful.

Imagine that Alice, a traveling consultant, visits one of her clients on a business trip. She has a laptop with built-in speakers, a smart phone, 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 or embedded displays.

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.

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 laptop display is actively being used by an email client. Alice accepts the system's suggestion and 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 for which she prefers private displays. She does not know where to find a private display on the airport, so she uses her smart phone to handle the task. The aggregation system, also running on her phone, finds out Alice's preferences from a recorded history of her past actions, discovers the closest devices (a keyboard and a display) that are available to be aggregated in the way that Alice likes, 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 automatically 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.

To address above requirements, we have designed and implemented the prototype of a system we call CAFE. CAFE stands for *Composition of Appliance Functionality in an Ensemble*, where 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 they are considered available for the purpose of performing a user's tasks.

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 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.

This paper is organized as follows. Sections 2 and 3 present the design goals and the architecture of the system. Section 4 describes the implementation and preliminary evaluation. Related work is described next. Section 6 presents conclusions and suggests possible future work.

2 CAFE System Model

In this section, we first define some common terms, then we discuss the key assumptions made by CAFE. Finally, we list out the requirements that CAFE aims to address.

2.1 Definitions

The following terms are used throughout the paper.

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.

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 [7, 2]. Examples include the user's task, the ensemble's devices, the user's location, and the time of day.

2.2 System Assumptions

Device 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.

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 automatically 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.

Infrastructure Support: We assume that all devices are connected with a shared, wireless network. We also 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 Metaglué [6]. 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.

2.3 CAFE Challenges/Requirements

CAFE aims to provide support for the automatic selection of a device aggregation that satisfies a user's preferences. Towards this goal, CAFE requires following specific capabilities:

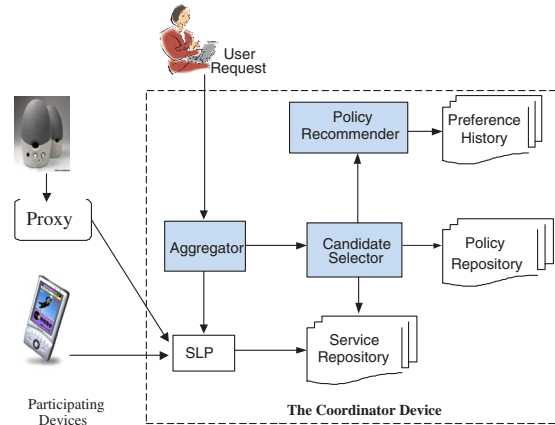


Figure 1: *The CAFE Architecture.*

1. User request needs be mapped to the required set of devices,
2. Selected devices should match the user preferences,
3. Device stability should be considered while selection to increase the aggregation stability, and
4. In case of new tasks, or any change in the ensemble, user distraction should be minimized.

3 CAFE System Architecture

In this section, 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 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 four components shown in Figure 1. The shaded boxes represent the run-time components that we developed, and the components inside the dotted box represent the entire CAFE system.

There are four main components of CAFE. **SLP Provider** registers and discovers services using Service Location Protocol (SLP) [11], **Aggregator** module calculates all possible aggregations for a given user task(s), **Candidate Selector** selects the aggregation among the candidates that best matches the user's preferences, and **Policy Recommender** predicts the user's preferences based on the user's past interactions with the system.

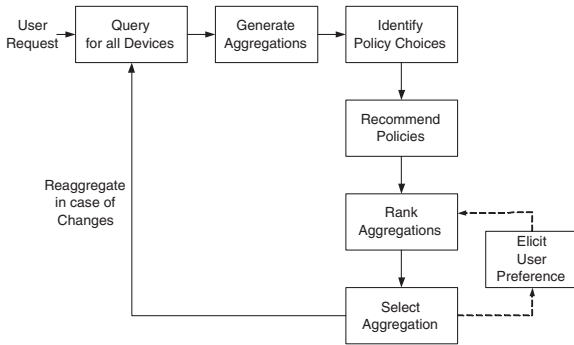


Figure 2: Steps involved in finding the best aggregation. Arrows with broken lines represent the optional flow, i.e. user preference elicitation is invoked only when the user wants to set the preferences.

3.1 Major Activities

Figure 2 presents the high-level flow of CAFE including interactions with the user and major computation steps. In addition to those steps, there is a system setup, which includes registration of devices with CAFE. We now discuss these major activities.

System Setup: Devices announce their availability and capabilities by registering with SLP. Registration includes location of the device, length of availability, and service description file. The latter is an XML-formatted document describing 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 run.

User Request: Task requests are specified as data-action directives. For example, to play the movie “The Matrix,” the user would specify the data source “TheMatrix.mpeg” and the action directive “play.” Similar directives are used

to express the services provided by different devices, and this helps CAFE achieve its first requirement (Section 2).

Query for All Devices: In this step, CAFE queries for all the registered devices from SLP. For each registered device, CAFE obtains the service description XML file.

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 [9]. 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.

Other Steps: Once the Aggregator generates all candidate aggregations, the Selector module is invoked to rank the aggregations according to the policies suggested by the policy recommender. Section 3.2 describes the policy mechanism that is used to rank the aggregations satisfying CAFE’s requirements. After aggregations are ranked-ordered, they can be returned to the user. As an alternative, the system can automatically instantiate the best candidate. If the user is not satisfied with the selection, she can set the policies to affect the policy recommender suggestions.

3.2 Scoring mechanism

CAFE uses multi-attribute utility theory (MAUT) [8, 5] as the scoring mechanism to rank aggregations. MAUT is an established technique in utility theory and has been used, for example, to model user preferences for product customization [4], and to evaluate security attributes for selecting among alternative security designs [3].

According to MAUT, the overall evaluation of a product can be defined as a weighted sum of its evaluation with respect to its relevant orthogonal value dimensions (attributes) [18]. We apply this mechanism to evaluating devices and aggregations. For scoring devices, the relevant value dimensions are the various properties such as

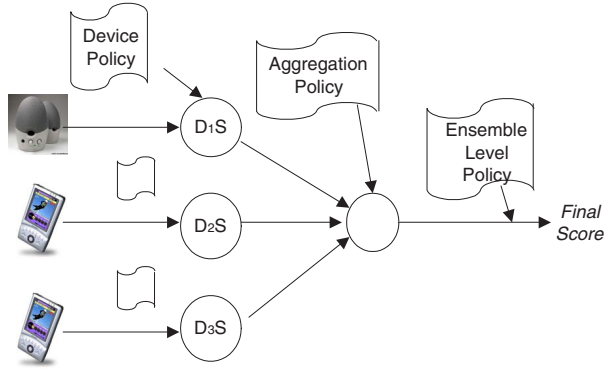


Figure 3: Scoring hierarchy. The final score of the aggregation is the weighted sum of the scores of the participating devices according to the aggregation policy. If this aggregation is the result of re-aggregation because of some change, the final score is adjusted to account for user distraction. This adjustment is applied in an ensemble-level policy.

size of the display, color depth, quality of sound. In case of aggregations, the relevant value dimensions are devices that are part of that aggregation. CAFE uses device stability and user distraction as additional dimensions for evaluating aggregations.

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 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 trade offs.

Figure 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 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.

3.2.1 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-

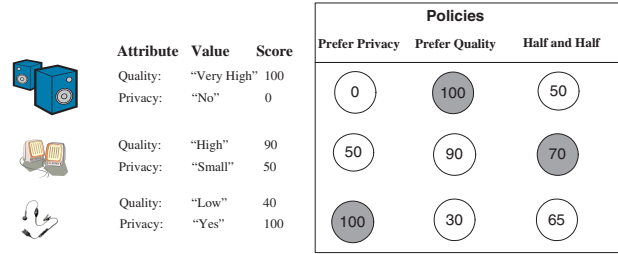


Figure 4: An example of scoring sound devices using a device scoring policy for playing sound when quality and privacy are the attributes being considered. The numbers in the circles give the score for the device after applying the service policy.

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 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:

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

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 gives an example of applying device scoring policies to compare sound devices.

The reason behind using addition function to score the devices lies in the fact that the device attributes are mutually *preferentially independent*, i.e. decision maker’s preference for the value of one attribute does not depend upon the values of other attributes. MAUT research shows that if the attributes are mutually preferentially independent, then the alternatives can be ranked using additive value functions [5].

3.2.2 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. The aggregation score is computed using additive value function similar to formula-1.

3.2.3 Stability and future planning

Once an aggregation is instantiated, changing it may inconvenience the user. Therefore, it might be worthwhile sacrificing 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.

The stability metric can be computed 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.

The stability factor is accounted while applying the aggregation policy. The scores of the individual devices are multiplied by their stability factor, and then aggregation scoring formula is applied to get the aggregation score.

3.2.4 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 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 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:

$$ADS(A', A) = \sum_{i=1}^d b_i(D'_i, D_i) * DDP_i \quad (2)$$

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 i th 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. 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.

$$ES_R(A') = AS(A', AP) - q_d * ADS(A, A') \quad (3)$$

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.

3.2.5 Adaptation to handle additional 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 an 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. In case of a conflict in device assignment, a task with higher importance is given preference.

3.3 Policy recommender

The policy recommender enables CAFE to closely 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 paper.

4 Implementation

In this section, we first describe the implementation of CAFE's components and then discuss our experience with using the system. 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.

4.1 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 1). These files include service descriptions, policies, and user history information.

4.2 Service Description and Registration

A device registers its availability by executing an HTTP POST of its service description XML file to the registration URL. Following 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 registration string for above mentioned *MPEG splitter* service is:

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

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 set based upon how volatile the environment is.

4.3 Aggregation Generation

For a given user request, the aggregator component performs the following steps:

- Query for all registered services from SLP
- 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

The JESS fact for *MPEG splitter* service is expressed as:

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

4.4 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.

4.5 Policy Suggestion

We used the implementation of the decision tree algorithm provided by WEKA [1] 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. 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.

4.6 Preliminary Evaluation

We performed experiments to demonstrate the functionality and to measure the performance of CAFE. In our experiments, we used CAFE to select aggregations for two tasks that a mobile user is likely to perform: playing an MPEG-formatted movie and editing a powerpoint presentation. We used prototype representatives of the devices to create virtual ensembles. For playing a movie, two devices are required: audio output and video output. For editing a presentation, three devices are required: input, video output, and audio output. We identified attributes for each type of device that we thought were relevant to our task needs. For example, for video output, we identified three attributes: size, color depth, privacy (CAFE infrastructure is flexible with respect to the number of attributes).

In the first scenario, we used CAFE to select an aggregation for one task: playing a movie. In the second scenario, we added the task of editing a presentation in addition to the first task of playing a movie. We experimented by changing policies, and by introducing devices with different properties along the given attributes. The experiments demonstrated the functionality of CAFE. In most cases, choices made by CAFE matched our intuition.

To demonstrate the efficiency with which CAFE is able to find and select an aggregation, we performed some experiments using an iPAQ 3635 running Linux as the coordinator device. Figure 5 shows CAFE’s average response time for different size ensembles for one user task. The time taken for finding all aggregations (*generate aggregations* step from Figure 2) shows the efficiency of running the JESS engine. The candidate selection time (*choose aggregation* step from Figure 2) 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. Since the policy files and JESS fact files were located on an externally mounted file system, the major portion of the response time corresponds to doing

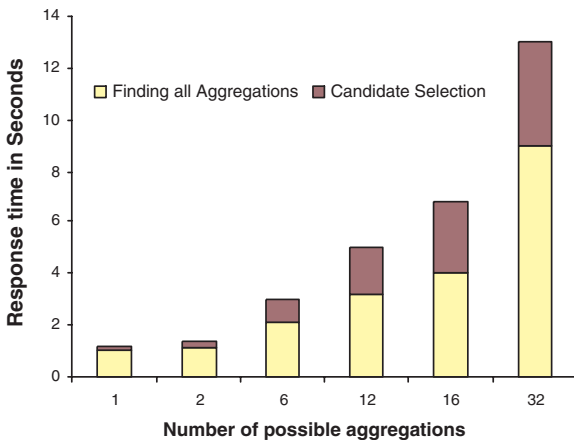


Figure 5: CAFE's response time for handling one task, averaged over 10 runs for every case. X-Axis shows the total number of possible aggregations for different ensemble sizes. Y-Axis presents system response times for the two main steps of CAFE's algorithm, i.e. finding all aggregations, and then finding the best candidate by ranking the alternatives.

i/o over PCMCIA card on the iPAQ. For the same experiment on a Pentium 1.1 GHz lap top, CAFE's average response times for above two steps were 133ms and 70ms respectively (in the case of 32 possible aggregations).

5 Related Work

Many projects have tried to provide a richer experience to the user in the presence of multiple consumer appliances [10, 12, 16, 17]. The basic idea that has been used is to represent devices as services, and then apply the techniques of service composition [15, 17]. 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 Metaglug 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 Metaglug describes the devices. The Rascal system, which is built on Metaglug, performs resource arbitration between multiple requests from multiple users in the context of an intelligent room [10]. 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. CAFE assumes wireless connectivity model to ease the constraint satisfaction and to improve the response time.

MAUT has been used by other projects [4, 13] to elicit user preferences in different contexts. 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.

6 Conclusion

This paper presented a system for automatic selection of devices for aggregation. We 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 provide users with a higher level of abstraction that helps users manage numerous aggregation choices. We also suggest that user's past actions and present context can be used to predict user preferences.

While our preliminary evaluation is encouraging about CAFE's usefulness and performance, user studies are necessary to evaluate the usability and functionality of the system more rigorously and objectively. 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.

In the near future, we would also 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.

7 Acknowledgments

The authors wish to thank the anonymous reviewers, WMCSA shepherd Trevor Pering, and members of the Agile team at HP Labs Palo Alto, whose comments helped improve the paper.

References

- [1] Weka 3 machine learning software in java: <http://www.cs.waikato.ac.nz/ml/weka/>.
- [2] G. D. Abowd, A. K. Dey, R. Orr, and J. A. Brotherton. Context-awareness in wearable and ubiquitous computing. In *Proceedings of the 1st International Symposium on Wearable Computers ISWC*, pages 179–180, 1997.
- [3] S. Butler. Security attribute evaluation method: A cost-benefit approach. In *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, pages 232 – 240, Orlando, Florida, May 2002.
- [4] D. N. Chin and A. Porage. Acquiring user preferences for product customization. In *Proceedings of 8th International Conference on User Modeling (UM-2001)*, pages 95–104, Sonthofen Germany, 2001.
- [5] R. T. Clemons. *Making Hard Decisions: An Introduction to Decision Analysis*. Duxbury Press, 1996.
- [6] M. Coen, B. Phillips, N. Warshawsky, L. Weisman, S. Peters, and P. Finin. Meeting the computational needs of intelligent environments: The metaglu system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
- [7] A. Dey. Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7, 2001.
- [8] G. W. Fischer. Multi-dimensional value assessment for decision making. Technical Report Engineering Psychology Laboratory Technical Report 037230-2-T, University of Michigan, June 1972.
- [9] E. J. Friedman-Hill. Jess, the java expert system shell. Technical Report SAND98-8206, Sandia National Laboratories, 1997.
- [10] 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.
- [11] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. IETF RFC 2608, June 1999.
- [12] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing Magazine*, 1(2), April-June 2002.
- [13] G. Linden, S. Hanks, and N. Lesh. Interactive assessment of user preference models: The automated travel assistant. In *Proceedings of 6th International Conference on User Modeling (UM-1997)*, pages 95–104.
- [14] D. Milojevic, P. Bernadat, R. Corben, I. Greenberg, R. Kumar, A. Messer, D. Muntz, E. O. Strain, V. Poladian, and J. Rowson. Appliance Aggregation Architecture(A³). Technical Report HPL-2003-140, HP Labs, Palo Alto, CA, 2003.
- [15] 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.
- [16] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: A service framework for ubiquitous computing environments. In *Proceedings of the Ubiquitous Computing Conference, Lecture Notes in Computer Science*, volume 2201, pages 56–75, 2001.
- [17] S. C. Samuel. Ninja paths: An architecture for composing services over wide area networks.
- [18] D. V. Winterfield and W. Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, 1986.