Functionality Adaptation: A Context-Aware Service Code Adaptation for Pervasive Computing Environments

Vivien Wai-Man Kwan, Francis Chi-Moon Lau, Cho-Li Wang
Department of Computer Science and Information Systems
The University of Hong Kong
{vjwmkwan,fcmlau,clwang}@csis.hku.hk

Abstract

Pervasive computing has attracted a lot of attention in recent years. Proxy servers that are suitable for pervasive computing have also been designed. They use different kinds of content adaptation techniques (such as distillation and transcoding) to adapt web contents in the content-rich servers to the resource-constrained devices. Although the adaptation of web contents have been widely discussed, less attention is put on the adaptation of services (or service codes), which is also important to enable computing anytime, anywhere, and on any device. In this paper, we present a proxy-based context-aware adaptation of service codes, known as functionality adaptation. *The main difficulty of such an adaptation is to estimate the resource usage required for the execution, which varies with the input size and is available only at run-time. We propose a conservative solution for the adaptation. A simple prototype has been implemented to evaluate the quality of the adaptation.*

1. Introduction

In the last few years, we have witnessed the proliferation in use of mobile devices and how mobile and pervasive computing [7] revolutionizes the way of computing. Today, computing is no longer limited to a particular location using non-mobile computing devices and can now be done on a heterogeneity of mobile devices, such as laptop computers and information appliances like Personal Digital Assistants (PDAs), smart phones, etc. Services can be accessed using these mobile devices wherever and whenever they want them: at home, in a meeting, or even when traveling. This change in the computing paradigm has made human lives easier and more convenient.

Due to the unique inherent characteristics of pervasive computing: computing device heterogeneity, limited device capability, and user's high mobility, information or services delivered in pervasive computing environments should provide *adaptability* support. A key to this is *context-awareness* [1]. Systems that are designed to provide services or information to the clients in pervasive computing environments should be context-aware, so that the services or information delivered can be adapted according to the target device's execution context, including the available resource of the device, location, time, and the behavior of the device user. Content adaptation, a common technique that has been used to adapt information to the client device according to the device capabilities, is an active research area that takes into consideration the contexts differences in mobile environment.

Although context-awareness issues is well explored with regards to information and web contents adaptation, the issues on the general application and service development process in pervasive computing environments are rarely discussed. In the past, traditional software applications, which target to be run in an environment with less context change, were written for only a limited number of platforms with few considerations on context-awareness issues. However, with the diversity of client platforms and contexts in the mobile and pervasive computing environments, it will be difficult for software developers to build applications that are able to handle different context scenarios for each of their targeted devices. On the other hand, although the introduction of Web Services, in which services are run on some dedicated servers, avoids the problem of running codes on heterogeneous platform, it is generally believed that running context-aware services (e.g. transcoding services) on the servers would put some extra burdens to the server, thereby making performance and scalability bottleneck to the services provided. Clearly, a proxy system is a suitable entity to offload the burden of the servers and provide context-aware services to the clients.

In this paper, we introduce a proxy-based approach for context-aware adaptation of service codes to the clients. Services are dynamically made up of mobile code compo-

nents, which are returned to the clients on demand. The adaptation makes use of run-time information provided by the clients to adapt service codes that can provide the desired functionality in the target device. This run-time information includes, among all, the resources available in the client device for executing the desired service or functionality. In order to determine service codes that are suitable for the clients, the proxy system needs to estimate the resource usage required for providing the service. This resource usage is not simply a static resource usage that can be known easily, but also includes the dynamic resource usage that depends on the execution and is only available at run-time. Despite the dynamics, this resource usage estimation needs to be available even before knowing the actual service codes that will be used by the clients at run-time for providing the service. Therefore, unlike traditional proxy servers, our proxy system does not simply act as a caching device, but also act as a broker and has the intelligence to select suitable service codes on behalf of the clients.

This paper is organized as follows. We first give an overview of the Sparkle Project, in which our proxy system is based on, in the next section; followed by the adaptation challenges and its adaptation principles in Section 3 and 4. Section 5 presents a simple prototype of the proxy system and its evaluation. Some of the related works are then given in Section 6, which is then followed by a conclusion in Section 7.

2. Overview of the Sparkle Project

Similar to other projects like Microsoft .NET [9], Sun Microsystems Open Net Environment [11], and the University of Washington's One.World [13], the Sparkle Project aims to build an infrastructure that is suitable for pervasive computing environments. However, instead of making use of Web Services, it supports downloading of mobile codes to client devices in an on-demand fashion. The infrastructure is based on the existing Internet infrastructure, with *adaptability*, *mobility support* and *peer-to-peer co-operation* as its main features. These three supports are important for infrastructures in pervasive computing environments. Figure 1 shows the architectural overview of the Sparkle system.

Mobile code components in our architecture are on-demand downloaded to the client devices, executed and then discarded. For example, it is possible to run an image processing application in a resource-constrained device to open, blur, find edges, or flip images, etc. An image processing application, *SparkleView* [10], was built to demonstrate the feasibility of our execution model. Requests are sent to the proxies at run-time for the necessary codes to be downloaded to the client device for execution.

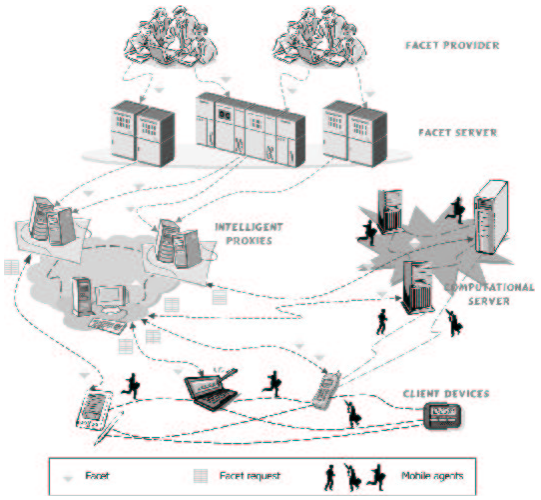

Figure 1. An architectural overview of the Sparkle system

2.1. Facets

Mobile code components that constitute services in our infrastructure are called *facets*. Each facet, when executed, provides a specific *functionality* for the clients. Functionality that a facet performs is regarded as a contract with clearly specified descriptions, inputs and outputs, together with pre-conditions and post-conditions. Users based on these contracts to decide upon the functionality they need. This functionality that a user requires, from the user point of view, is called a *service*.

In order to help adapting the code components, facets need to provide some information to describe themselves. A facet is, in fact, made up of two parts:

- **Code Segment.** This is the execution code that implements the functionality. It follows the contract and has one publicly callable method to be called upon by others. The client devices simply need to load this code segment in order to execute the specified functionality.
- **Shadow.** This is the metadata for describing a facet. It specifies the properties of a facet, such as the vendor, version, functionality, *facet dependencies*, and resource requirements, etc.

2.2. Facet Dependencies

A facet does not need to perform the whole functionality by its own. It might need other facets to help achieving its functionality. The services that a facet requires to help providing its functionality are called its dependencies. These

dependencies a facet requires can be represented by a *facet dependency tree*. For example, figure 2(a) shows a facet depending on the functionalities x, y and z. At run-time, these facet dependencies are used as requests for actual facets, which in turn have their own dependencies to help achieving their functionalities. For example, a facet for viewing an image requires a facet that can provide a functionality to decode an image. The proxy system chooses from among all those facets that can provide a functionality of decoding an image a facet to return to the client. This facet, in turn, might require another facet for providing some algorithmic functions to help decoding the image. These facets that are used at run-time to provide the image-viewing functionality to the client can be represented by a *facet execution tree*. An example of a facet execution tree is shown in Figure 2(b).

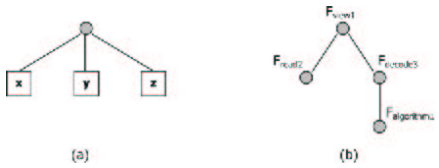

Figure 2. (a) A facet dependency tree and, (b) a facet execution tree.

3. Adaptation Challenges

The proxy system plays a very important role in supporting the downloading of facets to the client devices in an on-demand fashion. It has the intelligence to select suitable facets for the clients that can provide the desired functionalities. In order to make a suitable decision for the clients, the proxy system needs some information from the clients. *User registration* is also needed for adapting to device users.

The context-aware adaptation of service codes is not trivial. In particular, we have identified the following adaptation challenges:

- **Adaptation of Codes.** *Transcoding* is a usual practice in content adaptation. Some syntatic information is intentionally removed so as to reduce the resources it consumes but maintain its semantic meaning. However, codes are not for display, but for execution in the client devices. Any loss of the code structure during transcoding might result in an incomplete code that is unable to be executed. Even in a lossless transformation, it is difficult to ensure the transformed code can achieve the same functionality as the original code. In order to determine whether the two are equivalent during execution, semantic analysis of the codes, which implies human intervention, is required. This makes the adaptation difficult to be performed automatically on-the-fly. The situation is even worse if the codes are in executable format, where source codes are not available for semantic analysis.

- **Dynamic Configuration.** Static binding of components in service provision is not flexible for updating with new components. Instead of having a fixed binding between code components, dynamic binding is used in our model. Facets that constitute a service are dynamically bound when the corresponding functionalities are requested at run-time. This dynamic binding creates the flexibility for dynamic updating of services, but at the same time, also causes the facet execution tree unable to be known at compile-time. The proxy system is not even able to get an idea about the calling depth or the complexity of the tree. All these uncertainties increase the difficulty of making a selection decision for the clients.

- **Dynamic Resource Usage.** Unlike web contents whose resource usage can be determined statically, codes have dynamic behavior. Given a piece of code, different executions of the code may yield different results in the dynamic resource usages. For example, an image decoder uses 1MB of memory resources to decode a 800×600 image of 8-bit color depth; and 2MB when the image to be decoded is of size 1024×768 with a color depth of 16. This dynamic behavior of a code's execution makes it difficult to determine the amount of resources that would be used when it is to be executed in a heterogeneity of client devices.

4. Conceptual Design of Proxy System

In order to overcome these challenges, our proxy system makes use of a *two-phase adaptation* for selecting suitable service codes for the clients. The first phase, called the *filtering phase*, is to filter out facets that does not satisfy the requirements of the client. These requirements include, at least, the functionality needed by the client and the amount of resources available in the client device for executing the specified functionality. Facets that passes through the filtering phase are considered to have satisfied the client's requirements and are eligible for further processing. The second phase, called the *selection phase*, is to select a facet that best-suits the device user. This decision is based on the user preferences and other execution contexts of the client. The facet resulting from the two-phase adaptation is considered functionality-adapted and returned to the client. Figure 3 shows the processes involved in a two-phase adaptation.

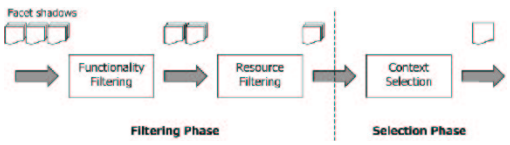

Figure 3. A two-phase adaptation.

4.1. Functionality Filtering

The first, and the most important, criterion for considering the suitability of a facet is the functionality it can provide. Facets that, when executed, cannot achieve the functionality required by the client are useless and should not be returned. With a functionality filter, facets that can provide the specified functionality can be selected and the number of facets to be processed in the later stages can be cut down significantly.

During the functionality filtering process, not only facets that are capable of providing exactly the same functionality is filtered. It is possible to have facets that can provide the functionality with a greater *capability*. For example, a facet that can decode images of size no more than 1024×768 is considered to be of a greater capability than one that can only process images of size no more than 800×600, since it can achieve what can be done by the latter and at the same time handle images of larger sizes. We say that these facets are of *compatible functionality* and should be able to pass through the functionality filter.

In order to determine facets of compatible functionality, special information is needed in the facet shadows for the proxy system to analyze. As mentioned before, facet shadow contains metadata information for describing a facet. Functionality is one of the information that needs to be provided. Different functionalities can be distinguished based on the description, input, output, pre-conditions and post-conditions. Table 1 gives the functionalities of two facets with compatible functionality.

Facets of compatible functionality have the same descriptions, inputs and outputs. However, the pre-conditions and post-conditions might be different. The capability of a facet, therefore, is specified by its pre-conditions and post-conditions. The pre-conditions specifies the input capability of a facet and the post-conditions specifies the output capability. Therefore, in functionality filtering, the proxy system tries to look for facets that have the same descriptions, inputs and outputs as the client's request, but pre-conditions and post-conditions that specify a greater capability.

The capability is represented by a range of values, called the *capability range*. In order to formulate a method for identifying a greater capability range, mathematical inequalities are used. Consider the two simple ranges:

Facet F_{decode_1}
Description: Image decoding
Input: A file containing the image to be decoded
Output: An array containing the decoded image
Pre-conditions: The size of the image to be decoded $\leq 800 \times 600$
Post-conditions: The image is decoded and put in the array
Facet F_{decode_2}
Description: Image decoding
Input: A file containing the image to be decoded
Output: An array containing the decoded image
Pre-conditions: The size of the image to be decoded $\leq 1024 \times 768$
Post-conditions: The image is decoded and put in the array

Table 1. Functionalities of two different facets.

$$5 < x < 10$$
$$0 < x < 15$$

It is not difficult to realize that every values in the first range is included in the second one, and the second range is said to be a greater range. So, if the first range in the above example is the range specified in the request while the second one is the capability range of a facet, we can conclude that the facet is of a greater capability than the one being requested. Therefore, in order for the proxy system to determine whether a facet is of a greater capability, it can make use of the *OR* operator in inequalities: if after applying the *OR* operation to the capability ranges of the facet and the request, and the resulting range is the capability range of the facet, then the facet is of a greater capability.

4.2. Resource Filtering

Apart from satisfying the functionality required by the clients, facets to be returned should also be able to execute completely in the client device. In this filtering process, facets that can satisfy the resource requirement of the client device are selected. This resource requirement is provided by the client and is the amount of available resources for executing the specified functionality in the client device.

In order to determine whether a facet can provide its specified functionality in the client device, the proxy system needs to be able to know the resource usage of the functionality that is provided by the facet. However, determining this resource usage is not an easy task. Recall that a functionality is not necessarily provided by a single facet, but can be provided by executing a number of facets.

Moreover, the resource usage of a facet can be divided into two parts: static and dynamic. Static resource usage of a facet depends on the code size, and can be known at compile-time, while dynamic resource usage depends on the execution as well as the inputs, which cannot be known until run-time. Being unable to have an accurate value of

the resource usage of a facet, a better approach might be to predict its maximum resource usage. This maximum resource usage can be treated as an upper bound, so that the facet can be safely executed and completed if the maximum resource usage can satisfy the resource requirement of the client device.

In predicting this maximum resource usage, the dynamics of the execution needs to be taken into account. Therefore, a fixed absolute or local maximum value is not preferred. A maximum value that varies according to different executions is needed. This value should vary according to the size of the input values that would be used for execution, and needs to be provided by the clients. If the same facet is used, the amount of resources used in an execution with a small input size should be expected to be less than that with a larger input size. Therefore, the proxy system needs a brief idea about the input size that would be used for execution by the clients.

On the other hand, each facet needs to provide a relation between the input size and the resource usage of the facet. This relationship is described by a *resource formula* in the facet shadow. A resource formula is a function of input size, i.e. $f(input\_size)$, so that the maximum resource usage for different input size can be estimated by the formula. For example, the dynamic resource usage of the matrix multiplication facet may be described by the formula $f(m_1, n_1, m_2, n_2) = 3(m_1 n_1 + m_2 n_2) + 2(m_1 n_2) + 10$, where m_1, n_1, m_2, n_2 are the dimensions of the two matrices. In order to allow individual facets to be easily developed without worrying about the resource usages of other facets that are required for execution at run-time, the formula only represents the resource usage of a single facet, i.e. a local resource usage.

In order for the proxy system to be able to estimate the resource usage of the functionality provided by a facet, the following assumptions are needed:

- The dynamic resource usage of a facet is represented by a resource formula.
- The dynamic resource usage of a facet increases with the input size.
- Each client request is accompanied by an approximate input size.

Such an estimation of the resource usage also implies a facet execution tree to be predicted for calculating the resource usage. The proxy system does not aim at predicting an accurate facet execution tree to be used by the clients at run-time, but the use of such a tree is only to help estimating the resource usage of the functionality. The prediction of a facet execution tree is based on the dependencies of each facet written in the facet shadow and are analyzed by the proxy system during the prediction. Instead of focusing on a single facet execution tree for predicting the resource usage, the resource usage predicted is based on all possible facet execution trees that can provide the desired functionality.

Among all these possible facet execution trees, the one selected by the proxy system is the one that uses the most resources for execution. The resource usage calculated is thus considered as the resource usage predicted for executing the functionality. In calculating the resource usage, the *worst-case input size*, which is the largest input size that would be used by the client for execution, is considered. With the worst-case input size, the resource usage of a facet execution tree is calculated as follows:

$$res(T) = res(F) + max(res(T_1), res(T_2), \ldots)$$

where F is the facet being the root of the tree, and $T_1, T_2, \ldots$ are its subtrees. Instead of summing all the resource usages of the facets involved, the maximum resource usage among all its subtrees is considered. This is due to the reason that the facets that have been used can be discarded by the client. Therefore, no two facets in different subtrees can be used at the same time.

Moreover, it is possible to have more than one facets that can provide the required functionality, of a different estimated resource usage. In that case, the resource usage for the functionality is determined by getting the minimum of them; i.e.

$$res(functionality_i) = min(res(T_{F_{i1}}), res(T_{F_{i2}}), \ldots)$$

where $F_{i1}, F_{i2}, \ldots$ are facets that can provide the desired functionality. Facets that can pass through the filter should be able to provide the desired functionality in the client device, and are candidates to be returned to the client.

4.3. Context Selection

In the selection phase, a candidate facet that is considered to best-suit the client's execution context is selected and returned to the client. This selection cannot result in a facet unable to perform the functionality in the client, but only how suitable it is for the client. In determining the suitability, the satisfiability of the device user is the main concern. The proxy system, therefore, needs some information about the device user in order to make a good decision that reflects the user's desire. This information that reflects the user's desire is the *user preferences*. It is a list of properties that the returned facets are preferred to have, and is usually stored in the *user profile*.

With this information available, selection is mainly base on the user preferences, such as vendor, version, etc. Users can also specify a resource usage level (high, medium, low) for the facets in their preferences. This resource usage level

indicates the *relative resource usage* among all the candidate facets for selection. For example, a high resource usage level indicates that facet that uses the most resources among all the facets that can provide the desired functionality is chosen. A low resource usage level is just the other way round. By allowing users to specify the resource usage level, they can control the relative resource usages of the facets that constitute a service. For example, a user, requesting for an e-mail service, would like to use more resources for security functions needed by sending the e-mail, and less for other kinds of functions.

Apart from the user preferences, the proxy system can also make use of other information, such as the status of the personal proxy cache and the user's past usage pattern, for better selection. They are information that can be stored in the proxy system and used as a kind of *proxy preferences* for selection. If a facet has been cached in the proxy, the time for returning it to the client can be less. This leads to a better performance that is likely to mean a higher preference for selection. In another case, if the usage pattern of a user indicates that a facet has been used before. It is also likely that the facet has a higher preference than the other candidates because they have ever satisfied the client (in another context). Actually, whether these facets are suitable to be given a higher preference depend on the proxy system. But these information might be able to act as some proxy preferences and help the proxy system in deciding the facet that is better for the client.

Each of the user and proxy preferences is given a score. Facets satisfying a user or proxy preference have the corresponding score added, and the one that scores the highest is returned to the client.

5. Prototype Implementation and Evaluation

As a proof-of-concept, a simple prototype of the proxy system was implemented. Context-aware adaptation of service codes is supported by making use of the run-time information provided by the clients, as well as information stored in the proxy system for better adaptation. Figure 4 shows the overall architecture of a proxy server.

In our prototype, facets are implemented as Java ARchives (JAR files), each contains the shadow and the code segment of a facet. The shadow of a facet is written as an XML file, while the code segment is a package of class files with one of the classes being the main class of the facet. Communication with the clients is done using Simple Object Access Protocol (SOAP) [12] while communication between the proxy servers is done by the socket interface. Requests received by the proxy servers are descriptions of functionalities and context information, written in XML format. In selecting a suitable facet for the client, the proxy system tries to match the XML request

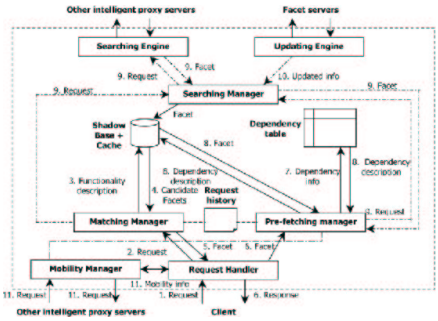

Figure 4. The proxy system architecture.

with the shadows that are also in XML format. Matching of the XML documents is based on XSet [6], an XML database and query engine from the University of California. The facet being selected is returned to the client, with the JAR file as an attachment to the SOAP message that briefly describes the facet.

5.1. Evaluation Metric

In order to evaluate the quality, we need a way to determine whether the adaptation achieved by our proxy system is good or not. Different from content adaptation in which some existing ways (e.g. analyzing the perceptual color depth, the percentage loss in transcoding, or the bandwidth consumption) are available to evaluate the quality of adaptation, there seems to have no existing evaluation mechanisms for evaluating the quality of functionality adaptation. Therefore, we need to define our own metric for the evaluation. This metric should reflect how well the service codes are adapted to the clients. A metric called *Adaptation Quality Index* (AQI) is, therefore, defined for our purpose:

$$\begin{aligned} AQI \quad = \quad & resource\_index \times weight(resource) + \\ & capability\_index \times weight(capability) + \\ & preferences\_index \times weight(preferences) \end{aligned}$$

where the three factors are assumed to have equal weights in our experiments and the three indices are calculated as described below. The resource index is a fraction of the resource usage in executing a functionality and the available resources in the client device. The capability index is a fraction of a facet's capability and the largest possible capability for the functionality. The preferences index is a fraction of a facet's score and the total scores of the preferences in the user profile.

5.2. Experiments

The prototype is tested on an Intel P4 2.26GHz PC (with Linux 2.4.18-3 installed) to examine the quality of function-

ality adaptation and its performance in processing a request from a client. The testing is based on a chess game called Othello. This game application is designed to use facets of 19 different functionalities. Instead of only providing 19 facets for the application, 5 facets of different capabilities are designed for each functionality to allow flexibility for the proxy server in selecting a facet fo the functionality. This makes a total of 95 facets available for the application.

In the experiment, requests with different requesting ranges and user preferences are used. Results show that different facets are returned by the proxy system, implying that the proxy system is able to adapt the service codes according to the client's execution context. Besides analyzing the facets returned, the AQIs of the facets returned are also calculated. The average AQI in adapting a facet is around $0.65 - 0.7$ (ideal: 1), i.e. around 65% of the ideal functionality. If variations in user preferences is ignored, the AQI can be as high as 0.85. Furthermore, the processing and decision times for selecting a facet to be returned to the client is also tested, and it turns out to be 300ms and 260ms on average respectively (figure 5).

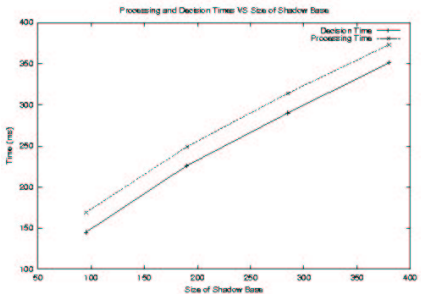

Figure 5. A graph showing the performance w.r.t size of shadow base.

We further compared our results with a random facet selection scheme. A request is sent to the proxy server for matching facets that can provide the required functionality. It then randomly chooses from among the candidates a facet for returning to the client, without trying to make better decisions. The average AQI in this case is about 0.59, and the average processing and decision times is about 198ms and 150ms respectively. Thereby, making the adaptation quality of our proxy system 160% better if we are willing to sacrifice half of the performance.

6. Related Work

The ability to adapt information and services to a diversity of computing devices is the key to pervasive computing. Adaptation can be performed in the server, client, or intermediary proxies. For flexibility, proxies are normally used in adapting the resulting contents before returning to the clients. However, these proxy systems usually only focus on adapting web contents for pervasive computing.

University of California Berkeley's TranSend [3] is an attempt to adapt web contents to heterogeneous client devices. *Distillation* is used to transcode the contents so that they can be handled by the resource-constrained devices. Distillation is a highly lossy, real-time, and data-type specific compression. Contents are distilled intelligently according to their data types, e.g. removing color information and formatting information that the devices are not able to understand. Quality is, thus, sacrificed to preserve most of the semantic contents.

Digestor [2] is software system that uses *automatic re-authoring* for adapting online documents. Re-authoring is the re-structuring of the documents such that they can be presented on the resource-constrained devices. Techniques for re-authoring includes outlining, first sentence elision, and image reduction and elision. Digestor makes use of an automatic re-authoring system to select the best combination of the re-authoring techniques for the document/display-size pair. The automatic system has a heuristic planner to help making re-authoring decisions, which is based on the heuristics captured in manual re-authorings.

IBM Transcoding Proxy [8] is also an http proxy that can transcode web contents for adapting to pervasive devices. It uses an *InfoPyramid* [4] as a data model to manage different *modalities* (e.g. text, audio, video) and fidelities (e.g. compressed image, summarized text) of the multimedia contents. and the transcoding methods for generating the different versions of the contents. *Translation* (change in modality) and *summarization* (change in fidelity) are the two major transcoding methods being used.

Transcoding is mainly used in these proxy servers for adapting web contents to resource-constrained devices. However, this technique is not possible to be used in adapting service codes. Proxy systems for adapting service codes have to use other approaches (e.g. selection) for adaptation.

Besides adaptation, the burden of locating services should be taken up by some intermediaries, such as agents or proxies. Users should only be responsible for describing the services that are required. The act of locating services are usually achieved by lookup services. In Jini [5], clients first use a discovery protocol to discover a lookup service, and then send their requirements to the lookup service. The requirements are specified in a *service template*, which is a structure-like data model indicating the search criteria. Services that match the criteria specified in the service template are returned. This is similar to the matching in the shadow base of our proxy system. However, Jini only aim at finding services for the clients and leaves the decision back to the

users. On the other hand, our proxy system also takes into account the user preferences and other information about the execution context, so as to intelligently select a suitable service for a client.

7. Conclusion

Pervasive computing is characterized by accessing information and services anytime and anywhere, through the use of small mobile devices. Moving around in this environment brings out the needs of customizing information and services according to different execution contexts. Proxy systems that are previously designed for adaptation to resource-constrained devices have only focused on adapting web contents. With the adaptation of service codes provided by our proxy system, clients are able to download service codes for execution, without worrying that the execution cannot be completed in the resource-constrained devices. This acts as a complement to the Web Services, and helps to enable truly pervasive computing by selecting service codes that are suitable for the clients to execute.

Conservative prediction has contributed to the core of functionality adaptation so that service codes selected by the proxy servers can complete the functionality in the client device. However, the prediction also has its own limitation in that requesters of the functionalities are required to provide a brief idea about the range of the input size to be used at run-time. Without this information, conservative prediction cannot be done and the proxy server is unable to determine whether the service codes can provide the functionality in the client device. Therefore, we hope that the followings can be done in the future:

- **Best-effort Prediction.** Although the requesters should be able to provide an approximation of the input size to be used, it is more flexible if the prediction also works when the size of the input data cannot be approximated. Despite that, the proxy server should still be able to predict the dynamic resource usage for executing a functionality and return service codes that can complete the functionalities in the resource-constrained devices. Without this information, the proxy servers might only be able to provide a "best-effort" prediction of the resource usage.

- **Proxy Server Modules.** At the current moment, the proxy server is assumed to run on resource-rich servers and have fixed connection to the Internet so that they can serve a larger number of clients at the same time. As peer-to-peer computing becomes more common, it seems to be a good idea if the proxy server can be made small enough to fit into resource-constrained devices. In that case, proxy servers can be run on small devices and be able to select suitable facets (although with some limitation) for nearby peers when the client is not connected to the Internet.

- **Improving the Evaluation Metric.** The current metric for evaluating the adaptation quality should have taken into account the important factors for the adaptation. However, there might still be some other factors affecting the adaptation quality. For example, the time for returning a facet to the client might be taken into account by the metric as well. A smaller downloading time (which implies a smaller static size and hence, resource usage) seems to favour. Apart from that, there might be other factors that can be added to the metric for enhancing the evaluation quality. Tradeoffs between different factors in a metric and should be carefully considered before adding them to the metric.

References

[1] 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*, 1997.

[2] T. W. Bickmore and B. N. Schilit. Digestor: Device-Independent Access to the World Wide Web. In *Proceedings for the 6th International World Wide Web Conference*, 1997.

[3] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *IEEE Personal Communications*, 5(4):10–19, 1998.

[4] C.-S. Li, R. Mohan, and J. R. Smith. Multimedia Content Description in the InfoPyramid. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1998.

[5] J. Waldo. The Jini Architecture for Network-Centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.

[6] B. Y. Zhao and A. Joseph. XSet: A Lightweight Database for Internet Applications. `http://www.cs.berkeley.edu/~ravenben/publications/saint.pdf`, 2000. Submitted for publication.

[7] IBM Think Research — Pervasive Computing. `http://www.research.ibm.com/thinkresearch/pervasive.shtml`.

[8] IBM Transcoding In Depth. `http://www.research.ibm.com/networked_data_systems/transcoding/In_Depth%/in_depth.html`.

[9] Microsoft .NET Homepage. `http://www.microsoft.com/net/default.asp`.

[10] Sparkle View. `http://www.csis.hku.hk/~clwang/projects/ipag-image.html`.

[11] Sun Microsystems Open Net Environment (ONE) Homepage. `http://wwws.sun.com/software/sunone/`.

[12] Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP/`.

[13] one.world Homepage. `http://one.cs.washington.edu`.