Challenges: Wireless Web Services

Hao-hua Chu, Chuang-wen You, Chao-ming Teng
Department of Computer Science and Information Engineering
National Taiwan University,
Taipei, Taiwan
{hchu@csie.ntu.edu.tw, r91023@csie.ntu.edu.tw, jt@csie.ntu.edu.tw}

Abstract

This paper describes the challenges of adapting existing web-service architecture to the wireless environment. It presents a new, wireless, web-service architecture based on the smart client model that can address some of the fundamental differences between the wireless and wireline environments. The fundamental differences between these environments can be called the mobile challenges, including (1) the unpredictable nature of the wireless network, (2) the limited processing capabilities and power on mobile devices, and (3) the need for consumer device to have a fast startup time for mobile applications.

The smart client model suggests the following modifications to the existing web-service stack in order to overcome these mobile challenges: (1) a lightweight web-service container that runs on a resource-limited mobile device (the client), (2) a quality of user experience (QoUE) model based on application response time, application startup time, and power consumption, and (3) an adaptive application configuration algorithm that can exploit the tradeoff among the QoUE parameters to provide the best user experience given preferences and the device platform.

1. Introduction

XML web-services are fast becoming the most prominent reusable, distributed, component-based technology with widespread acceptance from many industry players including: IBM, Microsoft, and HP.. Our definition of an XML web-service is similar to that of the W3C [12]. An XML web-service is identified by an URL, its public interfaces and bindings are described by WSDL, its service definition is published in an UDDI registry, and it can communicate with other web-service components using SOAP. A key benefit of the universal interoperability of web-services is that it enables vendors to construct new applications using a set of existing, reusable WSDL web-services. This application composition process can be very straightforward – in its simplest form, it consists of scripting the message flows among WSDL web-services. The composed application is called a business workflow, and the scripting language is called a workflow language. There are many workflow specification languages, such as BPEL [8] and WSCI [13].

This paper describes the challenges of constructing a new, wireless, web-service architecture, based on the smart client model The purpose of this architecture is to provide better end-user experience (i.e., good application response time, application startup time, and power consumption) when running mobile applications composed from WSDL web-services.

The technique of composing new applications from web-service components can be applied not just for building business flows, but also to build any future desktop or mobile applications. We define a *web-services-based application* as one that is created from components that are reusable WSDL services.

In the existing web-service architecture, web-services are intended to be run on the server-side rather than on the client-side. This results in the overall design of the web-service architecture, including its runtime modules (i.e., UDDI registry, workflow engine, and WSDL processor) being too heavy to run on a mobile device. *The existing web-service architecture inherently favors a thin-client model*, for supporting browser-based applications. In the typical lifecycle of a thin-client application, a client device first sends an HTTP request to an application server running remotely on the Internet. Upon receiving the request, an application workflow is instantiated either by static composition or dynamically composition (using multiple web-service components). The workflow is then executed in a workflow engine which controls the flow of interaction with the composing web service components. Finally it sends the response, in the form

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

of HTML, WML, or other presentation markup back to the client device. In the thin-client model, most of the computation occurs on remote servers, and a thin client must use the network and a server for application logic and data access.

1.1. Mobile Challenges

Relying on a more predictable network in the thin-client model can provide acceptable application performance in the wireline environment; wireless network connectivity, however, is unpredictable and can lead to poor application performance. The problem can be best understood by looking at the three fundamental differences between wireless and wireline environments.

Unpredictable Wireless Network: the wireless network is much less predictable than the wireline network. Wireless networks can suffer from network disconnections and fluctuating link bandwidth. Network disconnections are caused by moving outside the wireless network coverage areas, and fluctuating link bandwidth is caused by signal interference or fading. This network volatility causes problems for the thin-client model – applications become dead during network disconnections, and interactive applications have *unpredictable response times* in the presence of bandwidth fluctuations. The thin client model, when applied in an unpredictable wireless network, can result in a poor user experience.

Limited Computing and Power Resources: One obvious approach to overcoming this network unpredictability is the *thick client model*, where both application code and data are downloaded to the mobile client device for local execution. The problem with the thick client model is that a mobile device has limited computing (CPU, memory, flash) and power (battery) resources. For examples, a mobile device may not have enough memory to download and run an entire application, a mobile device may not be able to execute the application fast enough on its slower low-power processor, or a mobile device may not want to spend limited battery power to download application code and data where only a small portion is actually used and is only used for a short time. To address the limited computing and power resources on a mobile device, the thick client model needs to be revised to be selective and intelligent about which parts of the application should be executed locally on the mobile device and which should be executed remotely on the server. We call this model the smart client model. An important goal of the smart client model is finding the optimal application partition between local and remote executions according to user preference on power conservation and application response time.

Near-Instant Availability: Since mobile devices, such as cell phones, are considered consumer devices, mobile applications targeting them have different requirements than applications targeting desktop PCs. We would like to focus on one specific mobile requirement; *instant availability*. Instant availability means that mobile applications need to appear to be instantly available to a user, or the *application startup time* or *load time* needs to be fast. It is unacceptable to keep a user waiting, in order to download lengthy application code or data at load-time, before the user can start using an application.

Note that instant availability (application load time) forms an interesting tradeoff with application response time in the thin client and thick client models. Although the thick client model provides predictable application response time in the presence of variable network bandwidth, it can increase *application startup time* by requiring the mobile device to first download information as mentioned above. On the other hand, although the thin client model may provide unpredictable application response time in the presence of variable network bandwidth, it has a fast application startup time.

Our goal is to design a smart client model to provide *both* fast and predictable application startup and response time. There are several techniques that a smart client can utilize, such as *delay-download* or *partial-download*. When a user starts an application, the delay-download technique first applies the thin client model for fast application load time; at the same time, it also downloads code and data in the background. After the download is completed, the application then switches to the thick client model for improved application response time. The partial-download technique is used to identify a subset of application components for local execution. This subset can be downloaded and executed quickly on a mobile device to generate the first user interface screen for initial user interaction.

1.2. Quality of user experiences (QoUE)

Our presentd smart client model, called *WSmart*, considers the following three quantitative QoUE parameters.

- *Application response time (R)* is defined as the amount of time between a mobile device (e.g., a browser) receiving input from a user and that device providing a result (e.g., a web page) to that user.

IEEE COMPUTER SOCIETY

- *Application load time (L),* or *application startup time,* is defined as the length of time between a user starting an application and the first screen of the application's user interface being displayed.
- *Mobile power consumption (P)* is defined as the amount of power consumed by a mobile device to execute an application, which includes both power for processing and network transmission. Note that it does not include the amount of power consumed by remote servers, since we assume that remote servers have a comparatively large power supply.

The objective of WSmart is to optimize these QoUE parameters for a given application. Note that WSmart needs to take into account both *user preferences* and *mobile device heterogeneity*. Since different users can have different tolerances for the three QoUE parameters (e.g., a user in a hurry would want to want the best application response time even at a higher power consumption), WSmart needs to produce a different application configuration for different user preferences. In addition, since mobile devices have heterogeneous processing, memory and network capabilities, WSmart needs to find a different application configuration for each hardware platform.

1.3. Design tradeoffs

WSmart's design involves two important tradeoffs among QoUE parameters. The curve in Figure 1 shows the tradeoff between power consumption (P) and application response time (R) for a hypothetical application. It plots the points for a thin client, smart client, and thick client models on this *P-R curve*. In this hypothetical application, the thick client model performs mostly local execution on a mobile device; therefore, it is likely to consume higher mobile power, but it is also likely to deliver better response time since it requires little or no network communication. On the other hand, the thin client model relies mostly on remote processing; therefore it is likely to consume less power, but it is also likely to have longer application response time given it requires frequent communications with the server.

The curve in Figure 1 also shows the tradeoff between application load time (L) and application response time (R). It plots points for thin client, smart client, and thick client models on this *L-R curve*. In this hypothetical application, the thick client model requires lengthy downloads of code and data prior to execution; therefore, it is likely to have a long application load time given a slow wireless network.

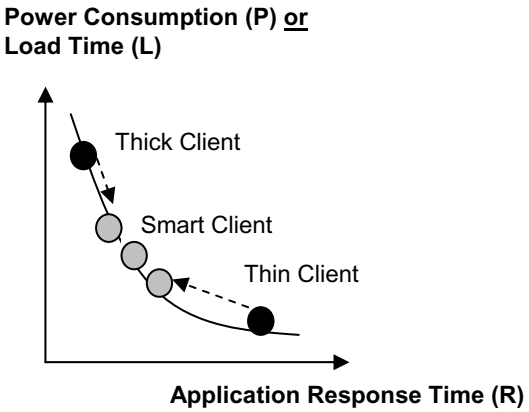

Figure 1. The curve represents the trade-off between mobile power consumption (P) and application response time (R) of a hypothetical application, as well as the tradeoff between application load time (L) and (R).

However, after the download is complete the application response time is likely to be fast, given that execution will have a decreased need to use the network. Conversely, the thin client model does not need to download any code or data prior to its execution; therefore it is likely to have fast application load time. However, the application response is likely to be poor since it requires frequent communication with the server.

The key point is that the smart client model can offer the *flexibility* to choose and move to any point along the curve between the thin client point and the thick client point at application load time and runtime. WSmart can configure a web-services-based application such that it offers the highest utility value given a user R-L-P preference.

1.4. Design principles

We would like to define three principles that guided the design of WSmart for web-services in a wireless environment. In order to differentiate these *wireless principles* from the *wired principles*, we will first describe the three wired principles given by Bosworth [1]. They are (1) *communication and server efficiency* allowing a server to scale up to a very large number of users and requests; (2) *loose coupling* between a messaging interface and the implementation, bringing interoperability between web-service components from different vendors; and (3) *asynchrony* with message-based architecture providing better support for server

IEEE COMPUTER SOCIETY

and network failures. Since wired web-service architecture is based on the thin client model as described earlier, these three wired principles mainly address issues on the server-side of web-service architecture.

In contrast, the wireless principles are mainly focused on the client-side of the web-service architecture. They are: (1) *Client efficiency,* achieved by running selected portions of application components locally on a smart client device. This is in contrast to a thin client configuration which is only capable of rendering a user interface. This also involves carefully managing the limited computing capabilities on a mobile device. (2) *Location-independent coupling* is the ability for a web-service component to be moved to the client side at runtime or load time, and continue to function with other web-service components running either locally or remotely. (3) *Dynamic reconfiguration* is adapting application configuration based on changing user preferences and heterogeneous device platforms.

We would like to use the following scenario to illustrate these design principles.

2. Scenario

Joe is taking a trip with his daughter Jane who has a WSmart-enabled mobile device with GPS and a slow 2G wireless network interface card. As Joe is driving on a highway with Jane, he notices an exit coming up. Joe is not sure whether he should take this exit or not, so he asks Jane for help to query a driving direction web-service. Since Joe may drive past the upcoming highway exit shortly, Jane requests the map service to provide the fastest possible startup time. As a result, the service adopts the thin client model, and is able to quickly provide the correct direction to allow Joe to take the upcoming exit.

As they drive into a busy downtown area where choosing the right streets to take becomes more complicated, they find that the direction service, currently running as a thin client, cannot provide fast enough response time to enable real-time GPS tracking and mapping. As a result, WSmart reconfigures the application from the current less responsive thin client configuration to a more responsive thick client configuration. In other words, it downloads application code and map data for faster local execution on Jane's mobile device. During the reconfiguration phase, the direction service continues to function without service interruption to Joe and Jane.

As Joe drives through the busy downtown area and onto another highway, he finds that application response time is no longer an issue. At the same time, Jane notices that the thick client configuration has consumed a significant amount of the battery life on her mobile device. In order to conserve the battery for this long trip, WSmart switches from power-hungry thick client configuration to a power-conserving thin client configuration.

It is near lunch time. Mary looks for directions to nearby restaurants by querying the direction service. Since battery life is equally important as application response time, WSmart switches from an all power-conserving thin client configuration to a smart client configuration which has a balanced mix of power conservation and response time. WSmart downloads a portion of application code and map data corresponding to the restaurant finder for fast, local execution. Since only a portion of the application code and map data is downloaded and executed on a mobile device, the smart client configuration may still need to communicate with the server but much less frequency than pure thin client configuration, resulting in sufficient (but not optimal) response time to guide them to a restaurant without consuming too much power.

3. Smart client model

To realize the above scenario, we define the WSmart architecture shown in Figure 2. WSmart can dynamically reconfigure application components for local execution on a mobile device, or remote execution on a server. It adds some additional components to the existing web-service architecture. We discuss these additional components and models in detail in the following subsections.

3.1. Client-side web-service engine

The client-side web-service engine is a lightweight engine that can execute web-service components on a mobile client device. The major components in a lightweight web service engine are a *k-flow engine*, a *k-UDDI registry*, and *k-SOAP/k-WSDL* processors. Due to the resource limitations of a mobile device, a lightweight engine will support a reduced set of functionalities of a full web-service engine. It will also introduce new mobile-specific functionality to support runtime reconfiguration of application flow.

One possible way to introduce mobile-specific functionality is to add new vocabulary to the web-service specifications. For the workflow languages such as BPEL, new vocabulary can allow a workflow author to specify all alternative, compatible and

IEEE COMPUTER SOCIETY

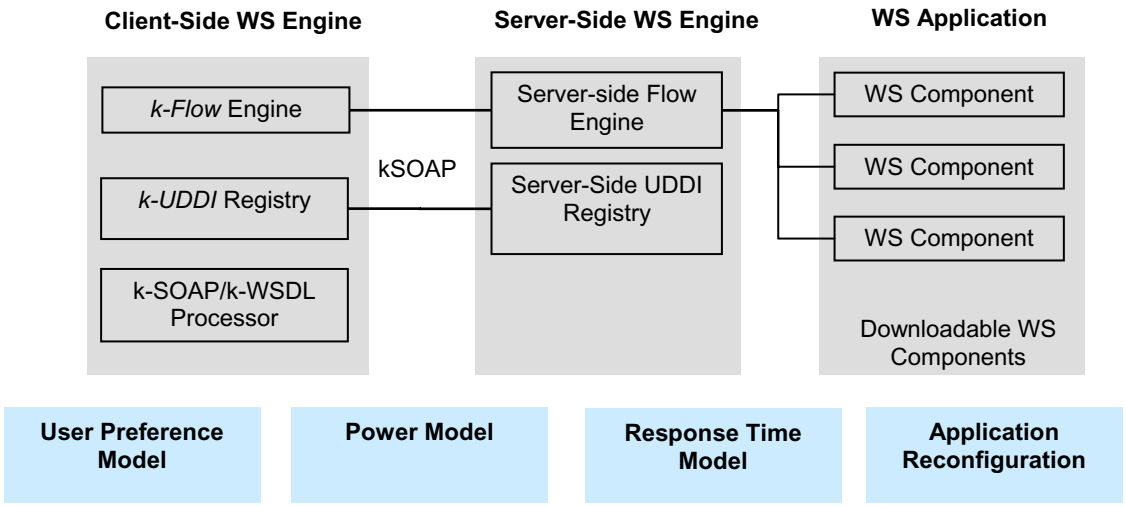

Figure 2. WSmart Architecture

runtime-interchangeable web-service components. For k-WSDL, new vocabulary can be added for a component vendor to publish whether its implementation (or fragments of its implementation) can be downloaded and executed on a mobile device. Additionally vendors can specify the estimated response time, load time, and power consumption for local and remote execution. For k-UDDI, the registry on the smart client only needs limited scalability to support services related to components of the executing workflow.

The lightweight web-service engine has two properties: *downloadability* and *configurability*. By downloadability, we mean functionalities provided by a lightweight web-service engine that can be expanded (or reduced) by downloading (or removing) additional (existing) engine modules. In other words, this is about a *highly customizable, configurable web-service engine* that accommodates different resource capabilities of heterogeneous mobile devices, and tailors itself to different requirements from applications. For example, a more capable mobile device can have a more feature-rich engine. In addition, if an application uses only a subset of engine modules, the other modules can be removed to make rooms for running applications on a mobile device.

By configurability, we mean that the processing can be split between a client-side lightweight web-service engine and a server-side full web-service engine. A logical web-service engine is split into two physical engines: one running on the smart client and one running on the server. The reason for this split is that the client-side flow engine can execute part of an application on the *performance critical path*, whereas the server-side flow engine can execute other parts of an application that are infrequently used. Downloadability and configurability in the engine can be used to optimize limited computing and power resources.

Research challenges are (1) what is the set of new vocabulary in the specifications and runtime modules to support dynamic reconfiguration? (2) How to design a lightweight, downloadable, configurable web-service engine?

3.2. User preference model

The user preference model can be represented as a utility function. This function takes three input parameters: application response time R, application load time L, and power conservation P. Since these three parameters have different units, a conversion is needed to normalize them into a common unit representing their relative importance to a user. This utility function can be represented as followed:

$$U(R, L, P) = w_R R + w_L L + w_P P$$

The weighting coefficients (w_R, w_L, w_P) in the utility function represent relative weights on R, L, and P, which can be derived from user preferences. A relatively high weight on one parameter means that a user considers it to be more important than other

IEEE COMPUTER SOCIETY

parameters. There are several possibilities for getting input from a user about these preferences: (1) request input manually from a user at the start of an application, (2) use a default policy specified by a user, or (3) infer from the current user context.

Note that user preference towards P can change based on the level of available battery power on a mobile device. If a mobile device is high on power, a user may worry less about conservation. Conversely, if a mobile device is low on battery, a user may have a strong preference for power conservation. This means that w_P is a function of the remaining battery power, rather than a constant value.

The research challenge is how to define conversion functions from (R, L, P) into one utility unit U.

3.3. Power consumption model

The power consumption model deals with the amount of battery power needed for a mobile device to access and run an application locally or remotely. Since power consumption for an application is the sum of power consumptions from its composing web-service components, it is necessary to model the power consumption at the component level. The power model can be represented by three power functions. (1) $P_{download}$ represents the amount of power to download a component for local execution. It requires mainly communication power to receive the downloaded code and data. (2) P_{lexec} represents the amount of energy to run a component locally after it has been downloaded. It consists mainly of processing power to execute. (3) P_{rexec} represents the amount of energy to access a component remotely. It is the power required to communicate with a remote server.

The power functions for the components are combined to form the power function of the composed application (P_{app}) as shown below:

$$P_{app} = \Sigma(P_{lexec} + P_{download} /(t_{session} / t_{download})) + \Sigma P_{rexec}$$

As described in the problem statement, the power functions ($P_{download}$, P_{lexec} , P_{rexec}) are dependent on the following dynamic wireless environment factors:

- Type and condition of wireless network (e.g., WLAN,GPRS, 2G).
- Hardware configuration of the mobile device (e.g., type of CPU, type of memory and flash).
- Application session, which is the duration of a session, is defined as $t_{session,}$ and the duration of a download is defined as $t_{download}$.

We can further express the P functions as parameters of these dynamic environmental factors. In addition, an application running on different networks or platforms can have different energy consumption. As a result, we need to have a mapping function that converts application resource usage (in CPU cycles, bytes transmitted and received, and memory/flash/disk access) into the amount of energy consumption for a given network and platform.

We can expect web-service component vendors to advertise their power functions. If the advertised power functions are unavailable, inaccurate or untrustworthy, we can apply runtime profiling techniques, or client-side monitoring of the actual power consumption. The monitored power values can be sent to a third party, for example the UDDI registry, where the advertisements are maintained and policed. Consequently, the advertised power functions can be updated according to monitored values.

The research challenge is how to define the power function/model at the component/application levels for local/remote execution. In addition, the power model should also consider issues such as network conditions, heterogeneous network handoff, and device platform heterogeneity.

3.4. Application response time model

An application response time is the sum of execution times from its components plus network communication latency. The response time model can be represented as two functions. (1) R_{lexec} represents the response time of a component when it is executed locally. It consists mainly of local processing time. (2) R_{rexec} represents the response time of a component when it is accessed remotely. It consists mainly of communication latency.

Note that the response time functions (R_{lexec}, R_{rexec}) are dependent on the following dynamic hardware factors, such as the type and condition of wireless network and mobile device hardware platform (i.e., speed of CPU). Response time functions may be supplied and specified by vendors. If they are not available or inaccurate, we can apply runtime profiling as mentioned in previous section. The response time functions for component are combined to form the application response function (R_{app}) shown below:

$$R_{app} = \Sigma R_{lexec} + \Sigma R_{rexec}$$

The research challenge is how to define the response time function/model at the component/application levels and for local/remote

IEEE COMPUTER SOCIETY

execution. In addition, the response time function should consider factors such as dynamic network link condition, heterogeneous network handoff, and platform heterogeneity.

3.5. Adaptive application configuration

Adapting application partitioning is done to optimize an individual user's utility function (derived from the user preferences) given an application power function, an application response time function, a set of components that make up this application, and the mobile device platform (i.e., the type of wireless link, the wireless link condition, and CPU speed). The core of the adaptive application configuration is an optimization algorithm that can compute assignments of either local or remote execution for each of the components of an application. The optimal assignments is a known NP-complete problem, requiring testing all the possible assignments. The time complexity increases exponentially with the number of components.

The reconfiguration process should not incur any noticeable service interruptions. For example, when a component is moved from a server to a mobile client, the runtime migration latency of components should avoid any visible degradation to the performance of the application. We will look at low-overhead, low-cost mechanisms for transparent component migration [4] between a remote server and a mobile client.

The power and computing overhead should be factored into the decision whether to reconfigure or not. If the reconfiguration cost is high, the benefit may not justify the cost. To minimize such overhead, we need to define good *trigger conditions* where significant changes in the wireless environment will cause the optimization algorithm to be run. For example, a trigger could be a vertical network handoff, a significant change in the wireless link bandwidth or energy consumption, or a user context change. In addition, a good trigger should avoid causing the application to constantly reconfiguring itself. There are other techniques that can be used to further minimize this overhead, such as (1) offloading the execution of the optimization algorithm to a remote server, and (2) applying computationally efficient heuristics to the optimization algorithm.

Some of the research challenges are (1) what are good heuristic algorithms to compute assignment of local or remote execution for each component that composes an application? (2) What are good triggers for running the optimization algorithm?

4. Related work

Satyanarayanan [10] outlines the fundamental challenges for mobile computing, to be resource-poor mobile devices, poor wireless network, and finite energy sources. He discusses the need for application-aware adaptation where applications work with the system during operation. Also discussed is extending the client-server model where some server functionality is moved to the mobile client device. Our work is an extension of his general design principle, applied to the domain of web-services examining the tradeoffs among QoUE parameters. Satyanarayanan [11], in another paper, defines fundamental challenges for pervasive computing which consists of a superset of mobile computing challenges. He mentions the client thickness issue, and discusses the advantages and disadvantages of thin or thick clients in a pervasive computing environment.

Banavar et al [3] has proposed a new application model for pervasive computing. They focused on the provision of pervasive information access from any devices, in any computing environment. They described techniques such as device-specific rendering, application apportioning, and application adaptation.

We classify the remaining related work into two main categories. The first category is based on compile-time analysis to decide how and where to split an application between client and server. This is done so that it can optimize system-level or application-level metrics, such as application response time, network traffic, and power consumption. The second category is focused on runtime techniques to reconfigure the application partition according to the changing values in these dynamic factors.

4.1. Compile-time partitioning method

Method partitioning [14] in JEcho (a distributed event system) uses compile-time analysis to divide a method into client and server parts. They represent each method using a control flow graph. Each node in the graph is an instruction and the cost of each edge is determined by a cost model. Splitting edges are determined using this method. Since each splitting edge incurs communication cost between the client and server, it finds an optimal set of splitting edges that minimizes communication cost.

The mutable services project [7] splits application components into edge and remote sets. Frequently used components are cached on edge servers that are closer to the mobile client devices. This helps to reduce

IEEE COMPUTER SOCIETY

application response time. In addition, they group multiple synchronous RMI messages into one large message between edge servers and remote servers. This helps to reduce the network latency of multiple synchronous RMI calls.

4.2. Runtime adaptation methods

Chroma [2] investigates remote execution on servers instead of on resource-poor mobile devices. It enables developers to specify different application partitions that take into consideration varying server resources in different mobile environments. These application partitions are called tactics. The Chroma system supports automated tactic selection based on the available computing resources at runtime, such as network bandwidth and server capabilities, as well as historically-based predictions on future application resource demands.

Coign [7] is an automated distributed partitioning system that constructs a graph model of the application's inter-component communication, and relocates strongly interacting components on the same machine to reduce communication overhead. It is based on a graph-cutting algorithm used to choose an application partition that minimizes execution delay due to network communication.

Agilos [6] discusses creating agile, adaptive middleware architectures to support application-aware QoS adaptation. One of the key considerations is an adaptor which monitors system-level states, such as CPU load or network bandwidth. The adaptor also promotes a global awareness of system-level resource availability. Other components are called tuner and configurator. They use the information provided by adaptors to adjust application parameters or activate functional reconfiguration actions based on a fuzzy control model.

4. Conclusion

This paper describes mobile challenges for web-services in a wireless environment. We present a smart client model called WSmart to overcome these challenges. WSmart leverages growing capabilities in computing resources on mobile devices to improve application response time, application startup time, and power consumption. WSmart is dynamic, reconfigurable and adaptive. It runs a lightweight, reconfigurable web-service container to locally execute select web-service components. It can reconfigure an application for local or remote execution.

There are related projects that address these mobile challenges such as application adaptation for a specific QoS parameter. WSmart, as presented here, addresses these challenges for a web-service architecture; in particular, it attempts to address the challenges as a whole, by understanding and exploring their tradeoff relationships, rather than addressing each individually.

We believe that the extensions presented to web-services architectures are an important extension to incorporate wireless environments.

6. References

[1] A. Bosworth, "A Conversation with Adam Bosworth", ACM Queue Magazine, March, 2003.

[2] R. K. Balan, M. Satyanarayanan, S. Park, T. Okoshi, "Tactics-Based Remote Execution for Mobile Computing", Mobisys 2003, May, 2003.

[3] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, D. Zukowski, "Challenges: An Application Model for Pervasive Computing", Mobicom 2000, August, 2000.

[4] H. Chu, H. Song, C. Wong, S. Kurakake, M. Katagiri, "Roam, A Seamless Application Framework", Journal of Systems and Software, Volumn 69(3), pages 209-226, January, 2004.

[5] G. C. Hunt, M. L. Scott, "The Coign Automatic Distributed Partitioning System", OSDI 1999, February, 1999.

[6] B. Li, W. Jeon, W. Kalter, K. Nahrstedt, J. Seo. "Adaptive Middleware Architecture for a Distributed Omni-Directional Visual Tracking System," MMCN 2000, pp. 101-112, January, 2000.

[7] D. Llambiri, A. Totok, V. Karamcheti, "Efficiently Distributing Component-Based Applications Across Wide-Area Environments", ICDCS 2003, May, 2003.

[8] OASIS, "Business Process Execution Language (BPEL)", http://www.oasis-open.org/.

[9] E. Pinheiro, R. Bianchini, E. Carrera, T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems", Proceedings of the Workshop on Compilers and Operating Systems for Low Power, September, 2001.

[10] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing." Fifteenth ACM Symposium on Principles of Distributed Computing, May, 1996.

[11] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges." IEEE Personal Communications, August, 2001.

[12] W3C Working Draft, "Web-service Architecture", http://www.w3.org/TR/2003/WD-ws-arch-20030514.

[13] W3C, "Web-service Choreography Interface (WSCI)", http://www.w3.org/TR/wsci/.

[14] D. Zhou, S. Pande, K. Schwan, "Method Partitioning - Runtime Customization of Pervasive Programs without Design-time Application Knowledge", 23rd International Conference on Distributed Computing Systems, June, 2003.

IEEE COMPUTER SOCIETY