

Extending Web Service Offerings Infrastructure (WSOI) for Management of Mobile/Embedded XML Web Services

Vladimir Tasic^{1,2*}, Hanan Lutfiyya², Yazhe Tang²

¹Department of Software Engineering, Lakehead University, Canada

²Department of Computer Science, The University of Western Ontario, Canada

vladat@computer.org, hanan@csd.uwo.ca, tyazhe@csd.uwo.ca

Abstract

Management of Extensible Markup Language (XML) Web services executing in mobile and/or embedded environments poses new challenges coming from limited available resources and from the need for characteristic management activities, such as handling context sensitivity. We determined management requirements, studied relevant tradeoffs, and designed a management infrastructure for monitoring of mobile/embedded XML Web services. The infrastructure enables that some or all management activities are executed on a non-mobile/non-embedded gateway instead of a mobile/embedded provider Web service. Further, it contains support for managing context-sensitivity and for handling disruptions in quality of service (QoS) and intermittent connectivity. While our high-level architecture can be implemented upon several existing Web service management tools, we chose to develop a new version of our Web Service Offerings Infrastructure (WSOI). The suggested solutions are being verified using the new WSOI prototype and validated on case studies, such as emulation of track-tracking Web services.

1. Introduction and Motivation

Mobile/embedded XML (Extensible Markup Language) Web services support ad hoc integration of diverse software running in mobile and/or embedded environments with other software running on the Internet, through the use of XML-based technologies such as World Wide Web Consortium (W3C) SOAP and Web Services Description Language (WSDL). Example application areas are mobile business, fleet management (e.g., truck tracking), and disaster relief.

We will use the truck tracking example [1] to illustrate important concepts. There is a frequent need

* V. Tasic was a Natural Sciences & Engineering Research Council of Canada (NSERC) post-doctoral fellow at the Univ. of Western Ontario. He was supported later by a Lakehead Univ. Senate Research Committee NSERC Research Development Fund grant.

to track movement and condition of trucks, particularly those transporting precious and/or hazardous goods. A mobile/embedded Web service may execute within a truck to provide information such as: geographic location, speed, quantity of gas left, air pressure in tires, and engine temperature. There are several possible clients of this Web service: the transportation company, companies whose goods are transported, police, and road assistance service.

Management of Web services is business-critical for many uses of Web services, because it ensures regular operation and handles run-time problems [2, 3, 4]. Management activities can be classified into monitoring and control. Monitoring includes measurement or calculation of quality of service (QoS) metrics (e.g., response time, throughput, availability), evaluation of requirements and guarantees, calculation of prices/penalties, and accounting. Control to meet guarantees and adapt to changes includes re-configuration of the Web service, re-negotiation of contracts used between Web services, and re-composition of Web services. Formal and precise specification of management information (e.g., in the form of Service Level Agreements – SLAs) is necessary for successful management activities.

Several management infrastructures for non-mobile/non-embedded Web services have been published [2-6]. Management of mobile/embedded Web services brings new challenges and specific requirements for management systems, elaborated later in this paper. Unfortunately, the existing Web service management infrastructures do not address these specific requirements. Various solutions [7-10] for handling context-sensitive operation, QoS disturbances, and/or disconnections for mobile/embedded devices are usually not directly applicable to mobile/embedded Web services since they deal with different mobile computing infrastructure and software.

We describe the first Web service management infrastructure for mobile/embedded Web services. This paper contains considerably more technical details

than our recent posters [11, 12]. The remainder of the paper is organized as follows. Section 2 summarizes requirements specific to management of mobile/embedded Web services and then Section 3 overviews achievements from related work. Section 4 describes a high-level architecture for the management framework based on the identified requirements. Section 5 describes implementation in an extension of our Web Service Offerings Infrastructure [2, 6]. The last section discusses conclusions and future work.

2. Specific Management Requirements

Requirements specific for management of mobile/embedded Web services [11] are additional to the requirements for non-mobile/non-embedded Web services, such as Web service discovery, Web service selection, composition verification, re-composition, and contract management. Security is also a common requirement for both non-mobile/non-embedded and mobile/embedded environments, so it is not discussed in this paper. (We recognize that the nature of the wireless medium makes that security issues are more important in mobile/embedded environments, but concentrate on management issues that are specific to these environments.) While we tried to be comprehensive, the following list of requirements might be appended in the future. Due to the space limits, we are not able to include justification examples in this paper, but they can be found in [13].

1. The system will support that providers, clients, and even management third parties can be mobile/embedded. While it will address specifics of management of mobile/embedded Web services, it will be possible to use it to manage, in a similar manner, any Web service and/or any Web service composition, irrespective of the execution environment.

2. It will be possible to distribute the system between mobile/embedded and non-mobile/non-embedded environments in such a way that the management functionality executing in mobile/embedded environments consumes minimal possible and relatively little processing power, memory, and electric power, because of the scarcity of these resources.

3. The additional communication over wireless networks for management purposes will be minimal, because wireless links are slow and unreliable and sending information consumes a lot of electric power.

4. The system will enable publishing, monitoring, storing, processing, analyzing, communicating, and updating information about the execution context (e.g., location) of a mobile/embedded Web service. It will also enable using context information for moni-

toring, accounting, billing, analysis, planning, and control activities in the management of Web services and their compositions. In the mobile computing literature, there are several different definitions of the term 'context'. A popular definition states "Context is any information that can be used to characterize the situation of an entity." [14] Unfortunately, management activities require precision that is missing from the cited definition. *Any* management information (e.g., values of QoS metrics) can be used to characterize situation of an entity, so it can be viewed as context. However, mobility brings new challenges to management systems, for which the management information maintained for non-mobile systems is not enough. Therefore, we adopted a stricter definition: *context* of an application refers to information about *external* run-time circumstances that influence execution [1]. A *context property* is an attribute of context. For a Web service, context does *not* include a description of its implementation, input values provided by a client, changes to its state caused by execution, or values of QoS metrics. For example, current time and country of location are context properties for a mobile banking Web service, but account balance is not (it is part of Web service's state). Similarly, the response time of code that implements a Web service is not context (it is determined by internal processes), but a network delay time for Web service messages is context (it is external and not fully controlled).

5. The system will provide mechanisms to adapt to changes in context of the mobile/embedded Web service. The adaptation activities include: a) re-configuration of the Web service, b) re-configuration of the management system (e.g., which management third parties are used), c) re-negotiation of contracts (e.g., which classes of service are used), and d) re-configuration of the Web service composition.

6. Since various disturbances in QoS can occur relatively frequently in mobile/embedded environments, the system will implement mechanisms for their handling with maximal possible efficiency and minimal possible overhead.

7. The system will maintain presence information about the Web service and provide this presence information to clients. This is useful for efficient management of Web service disconnections.

8. Since there is a possibility of relatively frequent disconnections of individual Web services, the system will provide application-level invocation retries (storing-and-forwarding of requests) to disconnected mobile/embedded provider Web services.

9. To handle disconnection and/or improve performance, the system will enable caching and pre-

fetching of results of operations of a mobile/embedded Web service, plus cache management (e.g., updates and invalidations) and lookup.

10. If several replicas of a Web service are available, the system will enable opaque exchange between them to accommodate disconnection of one of the replicas or achieve load balancing.

3. Related Work

Existing systems for hosting mobile/embedded Web services and systems for management of non-mobile/non-embedded Web services (e.g., [2-6]) *do not address* the requirements described in Section 2. The majority of tools currently advertised for “mobile Web services” actually enable only that clients execute on mobile devices, while providers are non-mobile. In such scenarios, provisioning and management challenges are significantly simpler than when providers are mobile/embedded. An important exception was (now obsolete) IBM’s Web Services Tool Kit for Mobile Devices (WSTKMD) – it enabled hosting of mobile Web services, but did not support their management. The Open Mobile Alliance (OMA) Mobile Web Services working group industrial standardization activity [15] is related mainly to hosting and security management, but it will not address all management challenges identified in our requirements. On the other hand, the Web Services Distributed Management (WSDM) [16] standardization activity and various existing management infrastructures for Web services [2-6] do not address the management issues characteristic for mobile/embedded Web services. Further, the run-time overhead of these infrastructures seems to be too high for mobile/embedded environments. Only WS-QoS [17] partially addresses some of these requirements.

While various solutions for handling context-sensitive operation, QoS disturbances, and/or disconnections of mobile/embedded devices were presented in the literature, in almost all cases these solutions are not directly applicable to mobile/embedded Web services. Most relevant are the works that discuss management of general mobile services (m-Services) that can use any distributed computing technology. For example, context management of m-Services was discussed in [7-9], handling of disconnections using software agents that store and forward information to disconnected mobile clients was presented in [9], while [10] researched caching of request results when mobile client is disconnected. However, these and similar papers focus on situations when only clients, and not providers, are mobile.

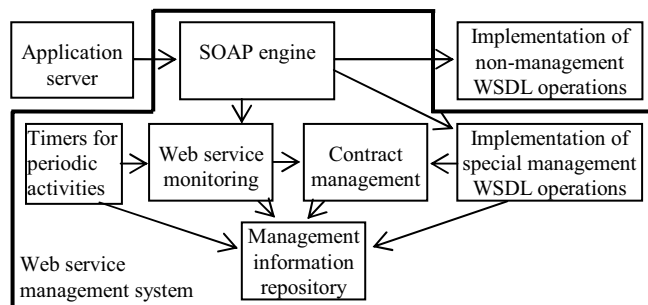


Figure 1. High-level architecture of our management system [12]

4. High-Level Management Architecture

The management functionality described in this section is based on the ability to extend the SOAP engine with monitoring of Web services, contract management, support for context sensitivity and handling of disruptions in QoS or intermittent connectivity. A SOAP engine refers to software that analyzes, processes, and generates SOAP messages. An example is Apache Axis [18]. Processing of SOAP messages is often important for management activities. Consequently, several Web service management tools (e.g., WSMF [4], WSOI [2, 6], Smartware [5]) were implemented as extensions of SOAP engines. In [12], we presented a high-level architecture of such a Web service management infrastructure, also shown in Figure 1. Hereafter, we focus on how we extended this general architecture to address the requirements specific to mobile/embedded Web services.

To deal with a limited amount of available resources (processing, memory, energy, bandwidth), the management infrastructure allows for the offloading of management activities from a mobile/embedded provider (client) to a non-mobile/non-embedded entity. This is based on our thorough preliminary study that concluded that in many situations, the majority of management activities, code and data structures that they use can be offloaded. Offloading requires several additional management parties: *gateways, proxies, and intermediaries*. We use the same terminology as [14, 12]. A *gateway* is a non-mobile/non-embedded entity that performs some message processing instead of the provider. It implements all WSDL operations advertised for the real provider, as well as specialized management functionality expected from providers. The client knows only the address of the gateway, not the real provider, and need not be aware what activities are performed by the provider and what by the gateway. Analogously, a *proxy* performs some activities on behalf of the client. *Intermediaries* process messages sent between the client and the provider.

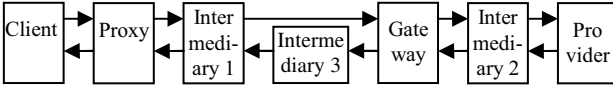


Figure 2. An example with multiple management parties

Relationships between these additional management parties, the client, and the provider are illustrated in Figure 2. The exchange of management information between management parties can be done using piggybacking into SOAP headers and/or using special management operations [2, 6]. The former mechanism consumes less bandwidth, but the exchange has to wait for the next SOAP message in which management information can be piggybacked. We analyzed how many gateways should be assigned to mobile/embedded web service provider: one in all provider contexts, several for different provider contexts, or one per class of service. In the current version, we allow only one gateway per provider.

Apart from providers, clients, gateways, proxies, and intermediaries, our architecture also supports probes, which periodically invoke the provider, e.g., to measure response time and/or availability. Since the use of probes can have drawbacks in business-to-business environments [2], we put more emphasis on intermediaries. All parties except providers and clients are optional – it is possible to combine parties into one entity, but this has both pros and cons [2].

Management activities require maintaining various information in the *management information repository*. Currently most web service management infrastructures provide values of measured/calculated QoS metrics, evaluated conditions, and billed prices/penalties. These are used as input into resource management, contract management, and Web service re-composition. For authentication and authorization of clients during session management, client information is needed. The management issues specific to mobile/embedded Web services require extensions of the management information model found in most existing management infrastructures (e.g., [2]).

To enable context-sensitive behavior of a mobile/embedded Web service, a management system should support storing, processing, and using context information. Several specialized formats for specification of context were developed for various mobile systems [7], but they are different from the models used for specification of other management information. As argued extensively in [1], there are many similarities (but also some differences) in both semantics and syntax of context properties and other management information, particularly QoS information. For example, both QoS metrics and context

properties have to be measured/calculated, aggregated, exchanged, processed, stored, and accounted in a similar manner. Using different formats to describe similar information can lead to problems in interoperability and increases run-time overhead, which is an important issue in mobile/embedded environments. *Describing context properties through extensions (subclasses) of models for other management information* is beneficial, so we adopted it in our architecture. Note that our architecture does not deal with measurement of context properties – we assume that such measurements come from some external modules, such as GPS systems for determining geographic location. However, complex context properties (e.g., “CountryOfLocation”) can be calculated from measured context properties in Web service monitoring modules analogous to those for QoS metrics. While we considered special support for presence, we decided to *model presence as a context property*. The main argument in favor of this solution is its relative simplicity. A client can get presence information from a gateway, so there is no need to wait for a timeout.

One way to differentiate service and QoS is to enable the provider to offer multiple classes of service to its clients. A *class of service* is a discrete variation of the complete service and QoS provided by a Web service. We use the term *service offering* to denote a formal representation of a class of service for a Web service [2, 6]. It is a special type of SLA, which is a special type of contract. Service offerings of one Web Service refer to the same WSDL description, but differ in management information, such as QoS guarantees and prices. The truck-tracking Web service could provide different service offerings, so that different clients (or the same client in different contexts) have access only a subset of operations and receive different granularity of provided information and QoS. For example, the transportation company needs information about the quantity of gas left, but the other clients should not have access to this information. Further, response time and precision of the operation returning truck’s geographic location could be high for the transportation company and the police and low for the insurance company. [2, 6] illustrated that providing and manipulating predefined service offerings is simpler, faster, and with less run-time overhead (in terms of memory, processing, and the number of exchanged SOAP messages) than providing and negotiating custom-made SLAs. These advantages are very beneficial in mobile/embedded environments. Consequently, our architecture supports service offerings.

Support for handling of disconnections using invocation retries, caching of results, and opaque ex-

change between Web service replicas also requires extensions of the management information model. While we developed algorithms that enable these three activities between any two parties, they are most likely to be used between a gateway and a mobile/embedded provider and between a proxy and a mobile/embedded client. Consequently, the corresponding information has also to be kept on the gateway and/or the proxy. For invocation retries, information about the disconnected party, retry timeout, retry interval, and maximum number of invocations has to be stored and used as input into a new type of periodic activities. This information need not be the same for all provider's operations. When the calling party gets a fault message that the called party cannot be reached (i.e., is disconnected), it schedules *periodic retries* until the provider is reached or timeout or maximum number of retries are reached. For caching of results, it is crucial to store both cached results of past invocations of the provider and detailed descriptions of when the cache is valid. These descriptions include cache expiration time and/or maximum duration, invalidating conditions (e.g., invocation of other operations), and information whether cached values can be reused between different sessions (e.g., different clients). As mentioned above, management information repositories of gateways and/or proxies contain long-term archives of invoked operations and their results. This information can be *reused as a cache*. Before contacting the mobile provider, the gateway looks up the cache and checks validity of found cache entries. If there was a valid hit, these results are returned without contacting the provider, thus saving time. The cache is updated whenever the gateway receives some results from the provider. Cache validity is updated during both the lookup and the update process. For opaque exchange between replicas, information about these replicas and their service offerings is needed. This information can be an expansion of information about alternative service offerings, so we modified algorithms and protocols for manipulation of service offerings to support switching between Web service replicas.

The *manipulation of service offerings* can also be used to handle disturbances in communication-level QoS, intermittent connectivity, and changes in context. For example, when these disturbances cause that guarantees in a service offering can no longer be met, then this service offering can be deactivated and clients switched to a service offering more appropriate to the new circumstances. Further, it is possible to use different service offerings in different context and specify conditions (constraints) to be evaluated on

change of context. For example, it is possible to define one service offering for Canada and another for the USA and define that whenever the value of the context property "CountryOfLocation" is updated, it is checked whether the current country is Canada. The results of these evaluations can trigger manipulation (e.g., switching) of service offerings that performs adaptation to the new context. In the previous example, when a truck with the truck-tracking Web service moves from Canada to the USA, the context property is updated, the condition is evaluated, and the switching from the service offering for Canada to the service offering for the USA is started. The main change in the previously published algorithms and protocols [2, 6] is that the gateway performs all activities previously performed by the provider, while the provider performs either only very simple initialization/finalization activities or even no activity at all. The analogous case is with the proxy and the client.

To support management activities, it is necessary to define *special management operations* that are invoked in management protocols. For our management system, we defined interfaces (collections of operations) for explicit session management, access right (security) management, providing information about available service offerings, comparison of service offerings, selection and manipulation of service offerings, push and pull exchange of management information between management parties, management of dynamic relationships between service offerings, and cache management. Only the last interface is motivated by management of mobile/embedded Web services, but it can also be used for non-mobile/non-embedded Web services. Since context information is modeled as a subclass of other management information, it can be exchanged in the same SOAP headers and/or management operations. The implementation of all management operations should be built into the management system, so that it can be reused between different Web services. The compatibility of management interfaces between non-mobile/non-embedded Web services and gateways (and proxies) used for mobile/embedded Web services supports uniform management of Web services executing in diverse environments.

5. Web Service Offerings Infrastructure (WSOI) 3.0

While our high-level architecture can be implemented upon several existing Web service management tools, we chose to develop a new version of our Web Service Offerings Infrastructure (WSOI) [2, 6].

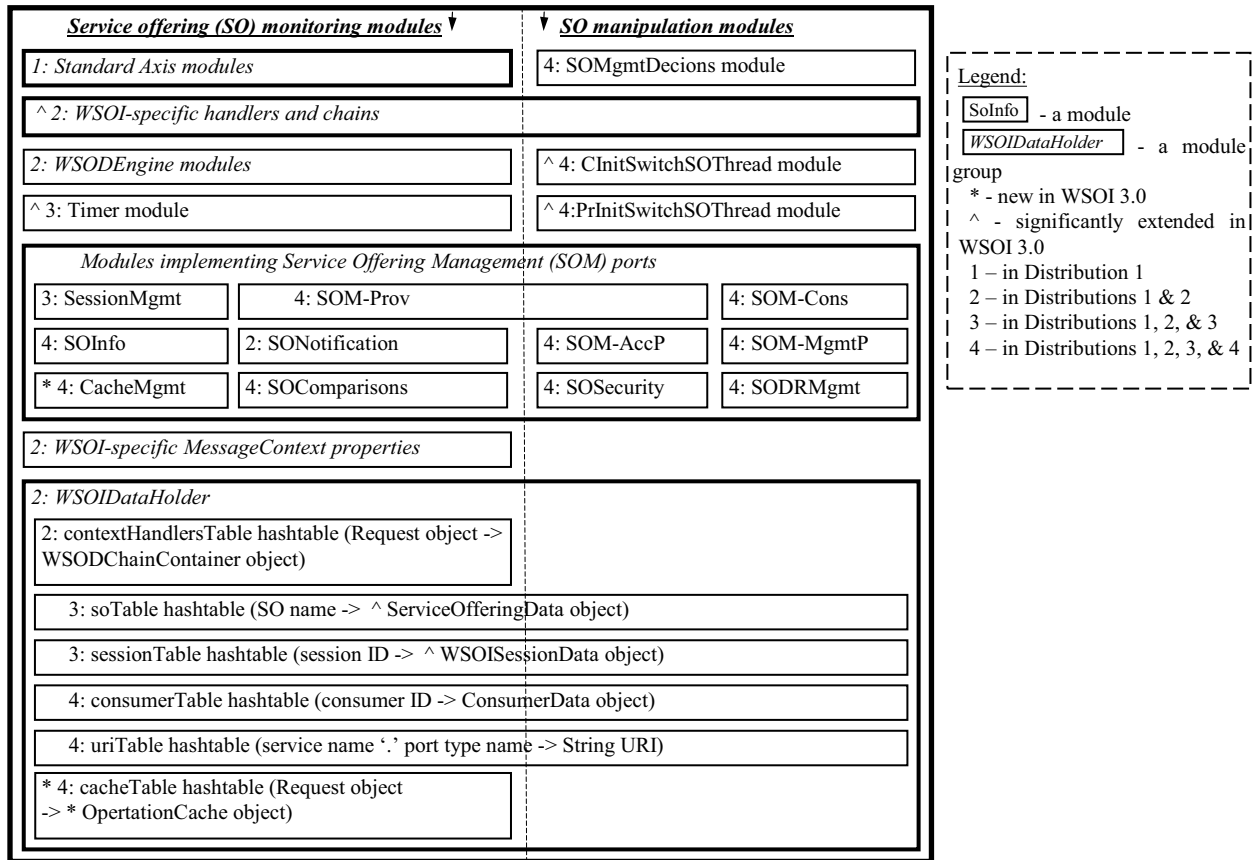


Figure 3. Modules and module groups in WSOI 3.0 Distributions 1 to 4

WSOI 2.0 was an extension of Apache Axis SOAP engine [16] and implemented (in Java) management solutions that are not specific to mobile/embedded Web services. For WSOI 2.0, service offerings containing descriptions of measured/calculated QoS metrics, evaluated constraints, and billed prices/penalties are specified in our Web Service Offerings Language (WSOL) 1.1 [6]. WSOL also enables specification of relationships between service offerings, which are used for manipulation of service offerings.

The new WSOI 3.0 adds support for management of mobile/embedded Web services, discussed in the previous section. We also developed the corresponding new WSOL version 1.4 [1]. Figure 3 shows the main modules in WSOI 3.0. It is an update of the figure given in [2] for WSOI 2.0. The modules on the left-hand side are used for monitoring activities, while those on the right-hand side (e.g., SOMgmtDecisions, CInitSwitchSOThread, and PrInitSwitchSOThread,) for manipulation of service offerings. The SOAP engine is contained in the standard Axis modules. Measurement/calculation of QoS metrics, evaluation of constraints, and calculation of prices/penalties are performed in WSOI-specific handlers and chains.

Handlers are Axis modules for processing SOAP messages, while chains are pipelines of handlers. WSOI handlers and chains are subclasses of Axis classes. The WSODEngine module is used for dynamic ordering of WSOI-specific handlers and chains for a particular invocation. The Timer module initiates periodic activities. Service Offering Management (SOM) ports implement specialized management interfaces. WSOI-specific MessageContext properties (specialization of Axis data structures) are used for short-term (per invocation) storage of values of QoS metrics, evaluated constraints, and billed prices and penalties. WSOIDataHolder management information repository stores all other management information and longer-term archives of monitoring results.

Figure 3 shows that only a few of the modules and classes (designated with “*”) in WSOI 3.0 are new and that they are related to cache storage and management. However, this is deceiving because new WSOI 3.0 features were mainly implemented through extensions of several existing WSOI 2.0 modules (designated with “^”). In particular, the WSOSessionData class is a complex data structure that archives information about invoked operations and per-

formed monitoring activities. It contains several Java Vectors of objects that are subclasses of the `MgmtInfoItem` class. Objects of `MgmtInfoItem` subclasses are also used by WSOI-specific `MessageContext` properties and several modules for SOM ports. WSOI 2.0 defined `MgmtInfoItem` subclasses for values of QoS metrics, constraints, and prices/penalties, while WSOI 3.0 added a subclass for context properties. In this way, context information can be stored, processed, used for monitoring and control, and exchanged between management party using mechanisms previously built into WSOI 2.0. Invocation retries are supported by extensions of the `ServiceOfferingData` class and new code in the `Timer` module. Apart from the new `cacheTable` hashtable (containing objects of the new class `OperationCache`) and the new `CacheMgmt` port, support for caching required modifications to the WSOI-specific chain `WSOIChain` to start cache lookups and updates. Modifications to the code for manipulation of service offerings to use gateways and proxies and to perform switching of Web services are in `CInitSwitchSOThread` and `PrInitSwitchSOThread`. They are supported by extensions of the `ServiceOfferingData` class.

Figure 3 also shows that WSOI 3.0 modules can appear in several distributions, as denoted by a number in front of module's name. This is the number of the simplest distribution where the module appears and implies that the module also appears in more complex distributions. WSOI 2.0 had only one distribution – all modules were present, even if they were not used. However, to reduce unnecessary overhead on mobile/embedded providers and consumers, we defined 4 distributions for WSOI 3.0 as most probable combinations of modules that might be needed. Distribution 1 supports no management activities (except implicit session management), so it can be implemented with any SOAP engine – only standard Axis modules are present. Distribution 2 supports per-invocation monitoring activities (except WSOL external operation calls) and does not perform periodic monitoring activities, management information archiving, and control activities. It adds several modules to Distribution 1, most notably WSOI-specific handlers and chains. Distribution 3 fully supports per-invocation monitoring activities, periodic monitoring activities, management information archiving, and session management (except security checks), but it does not support manipulation of service offerings. It adds several modules to Distribution 2, most notably the `Timer` module and substantial parts of the `WSOI-DataHolder`. Distribution 4 supports all management activities within the scope of our architecture and

contains all WSOI 3.0 modules. It is unlikely that this distribution would be used for mobile/embedded providers and consumers with limited resources, but it can be used for gateways and proxies.

To verify our solutions, we have been implementing a prototype of WSOI 3.0, based on our WSOI 2.0. The extensions of the management information model (i.e., classes related to `WSOIDataHolder`) and some modifications of manipulation of service offerings have been implemented and tested. For example, the tests showed that our approach to storing and processing of context information and using it in manipulation of service offerings is feasible. This means that the intended support for context-sensitivity and handling of disturbances in QoS is complete. The algorithms for cache management, invocation retries, and switching of Web service replicas have been designed, but are not yet fully coded and tested. We expect their completion in the near future. We have been using several case studies, such as the mentioned truck-tracking Web service, for validation of the determined requirements and the suggested solutions. At this time, we use only emulation of these case studies and not Web services executing in real mobile/embedded environments (e.g., trucks).

6. Conclusions and Future Work

In addition to monitoring and control activities relevant for all Web services, management of mobile/embedded Web services has to deal with limited resources (run-time memory, processing power, electrical power, and wireless bandwidth) and with characteristic management activities, such as handling context sensitivity, relatively frequent disturbances in communication QoS, and intermittent connectivity. The past research did not address management challenges specific to mobile/embedded Web services. Therefore, we identified requirements to address these challenges, studied relevant tradeoffs, defined general high-level architectural principles, and realized this architecture through the new WSOI version 3.0 and corresponding WSOL 1.4. We have been verifying the suggested solutions using a new WSOI 3.0 prototype and validating them on case studies, such as emulation of track-tracking Web services.

Our management architecture supports compatibility between management of mobile/embedded Web services and management of other Web services because it reuses and extends solutions, modules, and interfaces previously developed for non-mobile/non-embedded Web services. It deals with limited resources in mobile/embedded environments by off-

loading management activities to non-mobile/non-embedded management parties, particularly gateways and proxies. It models context information as a special case of other management information, and supports its storage, exchange, and use for monitoring and control activities using mechanisms same or similar to those used for QoS metrics. Presence information is handled as a context property. Invocation retries are implemented as a new type of periodic activities. Changes in context and disturbances in communication-level QoS are handled through manipulation of service offerings. Longer-term archives of invoked operations are used as a cache, along with new information about cache validity. Cache lookup (and invalidation, if needed) is performed before a gateway sends a request message to its provider, while cache update (and invalidation, if needed) is performed after the gateway receives a response message from the provider. Switching between provider Web service replicas is done as a generalization of switching between service offerings of the same provider.

It is important to emphasize that our management solutions are general in nature and not limited to the specifics of WSOL. In this paper, we concentrated on general concepts that can be adopted and/or adapted for other Web service management frameworks, particularly the WSLA Framework [3], WSMF [4] and Smartware [5]. This is because WSLA and WSMF use custom-made SLAs similar to our service offerings, while WSMF and Smartware are based on extensions of the Apache Axis SOAP engine.

Our short-term efforts focus on completing the WSOL 3.0 prototype. We also plan to validate our solutions on new case studies with Web services executing on real mobile devices, in addition to the truck-tracking case study that we emulate. Our longer-term objective is a powerful management system for Web services executing in diverse environments, both non-mobile/non-embedded and mobile/embedded. This work is an important step towards this goal.

References

- [1] V. Tomic, H. Lutfiyya, Y. Tang, "Web Service Offerings Language (WSOL) Support for Context Management of Mobile/Embedded Web Services", in *Proc. of ICIW'06* (February, 2006, Guadeloupe, French Caribbean), IEEE.
- [2] V. Tomic, B. Pagurek, K. Patel, B. Esfandiari, W. Ma, "Web Service Offerings Infrastructure (WSOI) – A Management Infrastructure for XML Web Services", in *Proc. of NOMS2004* (Seoul, Korea, Apr. 2004), IEEE, pp. 817-830.
- [3] A. Keller, H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", *J. of Network and Systems Management*, Plenum Publishing, Vol. 11, No 1 (Mar. 2003), pp. 57-81.
- [4] V. Machiraju, A. Sahai, A. van Moorsel, "Web Services Management Network: An Overlay Network for Federated Service Management", in *Proc. of IM 2003*, (Colorado Springs, USA, March 2003), IEEE, pp. 351-364.
- [5] A. Sharma, H. Adarkar, S. Sengupta, "Managing QoS through Prioritization in Web Services", in *Proc. of WISE 2003 Workshops* (WQW2003 at WISE 2003, Rome, Italy, Dec. 2003), IEEE pp. 140-148.
- [6] V. Tomic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma, "Management Applications of the Web Service Offerings Language (WSOL)", *Information Systems*, Elsevier, Vol. 30, No. 7 (Nov. 2005), pp. 564-586.
- [7] J. Indulska, D. De Roure "Proceedings of the First International Workshop on Advanced Context Modeling, Reasoning, and Management" (at UbiComp 2004, Nottingham, UK, Sept. 2004). At: pace.dstc.edu.au/ContextWorkshop2004Program.html
- [8] B. Benatallah, Z. Maamar (eds.) "Special Issue on m-Services", *IEEE Trans. On Systems, Man and Cybernetics, Part A*, Vol. 33, No. 6 (Nov. 2003), IEEE, pp. 665-757.
- [9] Z. Maamar, Q.Z. Sheng, B. Benatallah, "On Composite Web services Provisioning in an Environment of Fixed and Mobile Computing Resources", *Information Tech. and Mgmt.*, Vol. 5, No. 3-4 (July 2004), Kluwer, pp. 251-270.
- [10] D.B. Terry, V. Ramasubramanian, "Caching XML Web Services for Mobility", *Queue*, Vol. 1, No. 3 (May 2003), ACM, pp. 70-78.
- [11] V. Tomic, H. Lutfiyya, Y. Tang, K. Sherdil, A. Dimitrijevic, "On Requirements for Management of Mobile XML Web Services and a Corresponding Management System", in *Proc. of TELSIKS 2005* (Nis, Serbia, Sep. 2005), IEEE.
- [12] V. Tomic, H. Lutfiyya, Y. Tang, "A Management Infrastructure for Mobile/Embedded Web Services", in *Proc. of NOMS 2006* (Vancouver, Canada, Apr. 2006), IEEE.
- [13] V. Tomic, H. Lutfiyya, Y. Tang, "Major Requirements for a System for Comprehensive Management of XML Web Services", *Tech. Rep. #644* (Apr. 18, 2005), Dept. of Comp. Sci., Univ. of Western Ontario, 2005. At: flash.lakeheadu.ca/~vtomic/TomicEtAl-TechRep644.pdf
- [14] A. K. Day, "Understanding and Using Context", *Personal and Ubiquitous Computing Journal*, Springer-Verlag, Vol. 5, No. 1 (Jan. 2001), pp. 4-7.
- [15] Open Mobile Alliance, "OMA Web Services Enabler (OWSER): Overview", WWW page for *OWSER Draft Version 1.0*, Feb. 16, 2004. Web resource at: member.openmobilealliance.org/ftp/public_documents/mws/2004/OMA-MWS-2004-0011R02-OWSER-Overview.zip
- [16] H. Kreger, "A Little Wisdom about WSDM", *IBM developerWorks*, March 11, 2005. Web resource at: www.ibm.com/developerworks/webservices/library/ws-wisdom/
- [17] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller, "Efficient Selection and Monitoring of QoS-aware Web services with the WS-QoS Framework", in *Proc. of WI'04* (Beijing, China, Sep. 2004), IEEE, pp. 152-158.
- [18] The Axis Development Team, "Axis Architecture Guide, Version 1.0" Apache Axis WWW page, 2005. At: cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/architecture-guide.html