On Various Approaches to Dynamic Adaptation of Distributed Component Compositions

Vladimir Tosic, Bernard Pagurek,
Babak Esfandiari, Kruti Patel

OCIECE-02-02
June 2002

On Various Approaches to Dynamic Adaptation of Distributed Component Compositions

Vladimir Tosic, Bernard Pagurek, Babak Esfandiari, Kruti Patel
Network Management and Artificial Intelligence Lab
Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada
{vladimir, bernie, babak, kpatel}@sce.carleton.ca

ABSTRACT
We research dynamic reconfiguration of component systems in the context of dynamic adaptation. We classify various approaches to dynamic adaptation of component composition and conclude that they differ in power and complexity, yet still have significantly compatible strengths and weaknesses. Therefore, we argue for an integrated approach that implements several different approaches and leverages their compatible benefits by applying the appropriate one in a given situation. In the remainder of the paper we show how our recent research fits into this framework. This research is concentrated on Web Services as a specific type of distributed components. We explore Web Services with multiple classes of service and dynamic adaptation of Web Service compositions using manipulation of classes of service. Such an approach to dynamic adaptation is complementary to the reconfiguration of compositions of Web Services done by finding alternative Web Services and rebinding. Its benefits include adaptation speed, enhanced robustness of the relationships between components, simplicity, and low overhead. Also, providing multiple classes of service at the Web Service level has other benefits such as increased flexibility and choice for both Web Service vendors and consumers, while maintaining relatively low overhead and limited complexity of required management.

Keywords
Dynamic adaptation, Web Services, compositions of Web Services, component compositions, classes of service, service offerings, formal specification of constraints.

1 INTRODUCTION

Our research group has extensive experience in applying advanced technologies for managing computer and communication networks, distributed systems, and services. Several of our recent projects are related to the management of component-based software systems. At WCOP (Workshop on Component-Oriented Programming) 2000, we have presented our work on dynamic (i.e., run-time) service composition from components [11, 12]. At WCOP 2001, we have presented our work on software hot-swapping, i.e., dynamic software evolution with minimal system interruption [17, 4].

The theme of this workshop, WCOP 2002, is dynamic reconfiguration of component systems. In this paper, we present our resent research on dynamic adaptation of component compositions, particularly compositions of Web Services. In our work, reconfiguration is performed to adapt the component system to new circumstances and/or requirements. By 'reconfiguration of a component composition' we mean not only establishing new relationships between the composed components, but also modification of existing relationships.

After this introduction, we define a Web Service and discuss some of its distinguishing characteristics as a distributed component. Then, we argue for Web Services with multiple classes of service and present our language for formal specification of classes of service for Web Services. Next, we discuss in detail our work on dynamic adaptation capabilities using manipulation of classes of service. These capabilities are used for dynamic reconfiguration of component compositions, without breaking existing relationships between the composed Web Services. We also briefly present our ongoing work on an infrastructure supporting the concept of classes of service for Web Services and our dynamic adaptation management capabilities. After this presentation of our recent research, we classify approaches to dynamic adaptation of component compositions and advocate an integrated approach. We argue that our dynamic adaptation capabilities are a useful complement and addition to reconfiguration of component systems based on finding alternative components and rebinding. At the end, we summarize conclusions and challenges for future work.

2 WEB SERVICES AS COMPONENTS

Many leading computing companies—including Microsoft, IBM, Sun, Hewlett-Packard, Oracle, BEA, etc.—have recently announced their Web Service initiatives. These industrial initiatives are accompanied by the corresponding work of industrial standardization bodies, most notably the W3C (World Wide Web Consortium) [8, 3]. While definitions of a Web Service in different industrial initiatives vary somewhat, the common idea is that a Web Service is a unit of business, application, or system functionality that can be accessed over the Internet by using XML (Extensible Markup Language) messaging. Note that the goal of the work on Web Services is distributed application-to-application (A2A) and business-to-business (B2B) integration—particularly ad hoc, impromptu, and temporary in nature—over the Internet. Consequently, the true power of this technology is not in providing stand-alone Web Services for human users, but in composing Web Services provided by independent business entities, potentially imple-

mented in different programming languages, and potentially running on different platforms.

Web Services are components, as defined in the call for papers for this workshop [2]. That is, Web Services are independently-deployed units of third-party composition with explicitly specified contractual interfaces. Often, this composition is performed dynamically, i.e., during runtime. However, Web Services are not software components, as defined by [16] and [6]. One of the differences is that Web Services—contrary to software components—are not units of independent deployment by the composing parties. Web Services are already deployed over the Internet and the composing parties only use them. Further, Web Services are not binary units of independent production because their implementation is hidden. One binary unit of program code (e.g., a software component) can implement several Web Services. In addition, a Web Service can, in principle, provide not only software functionality and data, but also access to some hardware resources such as memory, printing, network bandwidth, etc. In spite of these and other differences, the similarities between the two concepts make a significant part of the work on software components relevant for Web Services, and vice versa. While our work is oriented towards Web Services, we find it very relevant for the theme of this workshop.

3 WEB SERVICES WITH MULTIPLE CLASSES OF SERVICE AND THEIR SPECIFICATION

As the number of Web Services that offer similar functionality increases in the market, the offered QoS and price/performance ratio, as well as adaptability, will become the main competitive advantages. In certain circumstances, it can be useful to enable a Web Service to offer several different classes of service to consumers. Hereafter, by a consumer of a Web Service *A* we assume another Web Service that is composed with *A* and collaborates with it, not an end user (human) using *A*. One Web Service can serve many different consumers, possibly at the same time.

A class of service is a discrete variation of service and QoS (Quality of Service). Classes of service can differ in usage privileges, service priorities, response times guaranteed to consumers, verbosity of response information, etc. The concept of classes of service also supports different capabilities, rights, and needs of potential consumers of the Web Service, including power and type of devices they execute on. Further, different classes of service may imply different utilization of the underlying hardware and software resources and, consequently, have different prices. Due to the flexibility to accommodate several classes of consumer, providing different classes of service and their balancing helps vendors of Web Services to achieve maximal monetary gain with more optimal utilization of resources. On the other hand, consumers of such Web Services get additional flexibility to better select service and QoS that they will receive and pay for and minimize the price/performance ratio and/or the total cost of received services. Additionally, different classes of service can be used for different payment models, such as pay-per-use or subscription-based. While providing differentiated services and multiple classes of service are well-known concepts in telecommunications [9, 1], we are trying to address issues relevant for Web Services and their compositions, including dynamic adaptation of compositions of Web Services using manipulation of classes of service. Studying issues related to classes of service at the Web Service level has practical relevance for Web Service compositions, but it has not been addressed by prior works.

We have conducted a thorough analysis to compare service differentiation using classes of service with relevant alternatives, including parameterization, multiple ports, multiple Web Services, personalization techniques such as user profiling, etc. The main advantages of having a relatively limited number of classes of service over other types of service differentiation are limited complexity of required management and relatively low overhead incurred. For example, we find that personalization techniques aimed at human users can be too complex for simpler Web Services composed with other Web Services. We want to limit the complexity and overhead in order to assure solutions are scalable to large compositions of Web Services. In addition, classes of service are supported by many underlying telecommunications technologies. Our approach is an additional and complementary mechanism for discrete differentiation of service and QoS, and not a complete replacement for alternatives. It might not be appropriate for all circumstances, e.g., due to its own overhead. Later in this paper, we argue for an integrated approach to dynamic adaptation of compositions of Web Services, where one of the techniques is based on the manipulation of classes of service.

One of the conclusions of our past project on dynamic service composition [11, 12] was that comprehensive formal specification of components supports selecting appropriate components in the process of dynamic service composition and that it can help reduce unexpected interactions between the composed components. Consequently, we need a comprehensive formal specification of Web Services with multiple classes of service.

We define a service offering as a formal and unambiguous representation of one class of service of one Web Service or one port of a Web Service. Service offerings of one Web Service relate to the same characteristics described in the corresponding WSDL (Web Services Description Language) file, but differ in constraints that define classes of service. These service offerings are specified separately from the WSDL description of the Web Service. They form electronic contracts between the composed Web Services and are a basis for monitoring and management activities. Hereafter, we will occasionally use the shorter term ‘offering’ with the meaning ‘service offering’.

A port-level service offering specifies only constraints upon the constructs in the referred port. A component-level service offering of a Web Service with multiple ports de-

scribes an allowed combination of port-level offerings. If a Web Service has only one port, the component-level offering is identical to the corresponding port-level offering.

We specify service offerings for Web Services in a comprehensive XML-based notation called WSOL (Web Service Offerings Language). More information about WSOL and the status of its development can be found in [19], while here we will give only a very brief summary. The syntax of WSOL is defined using XML Schema. WSOL is a fully compatible extension of WSDL (Web Services Description Language). WSDL is a W3C standard for describing Web Services in an XML notation. However, WSDL does not enable specification of various constraints on operations and ports in a Web Service. While WSDL can (and has to) be extended in several different areas, WSOL extends WSDL only with capabilities directly relevant to the concept of service offerings and the related formal specification of various types of constraints.

In WSOL, specifications of different constraints are separated into distinct constraint dimensions to achieve greater flexibility and reusability of specifications. This is a separation-of-concerns issue. However, to support easier choice by consumers, these constraint dimensions are integrated into a service offering.

WSOL currently enables formal specification of:

- functional constraints (pre- and post-conditions),
- QoS (a.k.a. non-functional) constraints,
- simple access rules,
- price,
- relationships with other service offerings of the same Web Service, and
- information about which entity (the Web Service, the consumer, or some trusted third party) is responsible for monitoring particular constraints.

QoS constraints describe properties such as performance, reliability, availability, etc. Note that access rules in WSOL describe what subset of Web Service's operations a service offering allows consumers to use, i.e., they serve for differentiation of service. Conditions under which particular consumers or classes of consumer may use a service offering are specified and stored outside the WSOL description of a Web Service. Specification of relationships between service offerings is important to support: 1) easier and more straightforward specification of relatively similar service offerings of one Web Service or several similar Web Services; 2) easier discovery, selection, and negotiation of appropriate service offerings; 3) capabilities for dynamic adaptation using manipulation of service offerings. These capabilities will be presented later in this paper.

In the future, we will make specification of the constraints currently supported in WSOL more powerful. We also plan to add formal specification of some other constraints. Further, we are working intensively on proof-of-concept prototypes [19] for a WSOL parser, an automatic generator of constraint-checking Java code, and a Java API (Application Programming Interface) for generation of WSOL files.

We are still working on some issues related to the separation and integration of constraint dimensions. Another issue that we are re-examining is specification of relationships between service offerings. Our current solution is based on constraint dimensions. Yet another issue is definition of formal and unambiguous ontologies for different QoS metrics, measured properties, measurement units, currencies, and other terms related to monitoring and management of Web Services. For example, the QoS metrics that are used to define QoS constraints have to be ontologically defined because they can be easily misinterpreted. It is important that these ontological definitions include specification of dependencies and relationships between QoS metrics. In [20], we have discussed requirements for such ontologies, as well as why existing ontologies are not appropriate.

WSOL is, as far as we know, the only ongoing work on formal specification of classes of service for Web Services. On the other hand, there are several other ongoing works on formal specification of various types of constraints for Web Services. One example is DAML-S (DARPA Agent Markup Language – Services) [4]. We have compared WSOL with several such efforts in [20]. Further, in [19] we have related WSOL to the previous works on formal specification of various constraints, and particularly such works using XML.

More information about WSOL can be found in [19]. This additional information includes an illustration of the WSOL concepts with an e-business example, an illustration of the WSOL XML schema with an example service offering in WSOL, and a brief overview of prototypes of WSOL tools.

4 DYNAMIC ADAPTATION OF COMPOSITIONS OF WEB SERVICES WITH MULTIPLE SERVICE OFFERINGS

Composing complex systems from Web Services, especially during run-time, can significantly increase system agility, flexibility, and adaptability. However, to further increase these qualities, such compositions have to be managed and adapted to various changes, particularly to those changes that cannot be accommodated on lower system levels such as communication software, operating system, etc. This management and adaptation should occur while the system is running, with minimal disruption to its operation and with minimal human involvement. In other words, it should be dynamic and autonomous. Dynamic composition of Web Services is out of scope of the work presented in this paper. We assume that Web Services are already composed into a component system, and address only dynamic adaptation of this composition of Web Services.

We want to achieve dynamic adaptation of compositions of Web Services without breaking an existing relationship between a Web Service and its consumer. This goal differentiates our work from the past work on architecture-based adaptation approaches based on finding alternative compo-

nents and rebinding [14, 13, 15, 10, 6, etc]. To achieve this goal we are researching dynamic adaptation capabilities based on manipulation of service offerings. Our dynamic adaptation capabilities include switching between offerings, deactivation/reactivation of existing offerings, and creation of new appropriate offerings. These capabilities are under control of the Web Service and therefore their use, especially the creation of new service offerings, can be restricted. A vendor of a Web Service can decide that under some circumstances, or permanently, some classes of consumer cannot use some of these capabilities in a particular Web Service due to security reasons, because these capabilities are not fully implemented or other reasons. An e-commerce example illustrating these three dynamic adaptation capabilities is given in [19].

Note that one of the crucial issues related to these capabilities is how to appropriately relate service offerings. Some of these relationships are essential in deciding on which offering to switch (particularly after deactivation of some offering), while others are necessary for efficient and correct dynamic creation of new offerings. We address this issue in our work on WSOL, where we pay particular attention to such issues relevant for dynamic adaptation and management.

Dynamic Switching of Service Offerings
Switching between service offerings is equivalent to changing which offering a consumer uses. It is the basic method for dynamic adaptation in our work. Switching can be initiated by a consumer or by the Web service. This capability enables consumers to dynamically adapt the service and QoS they receive without the need to find another Web Service. It also enables Web Services to upgrade or degrade their service and QoS seamlessly. Switching should be done only in a way that constraints in the new service offering are not violated. The Web Service should prevent any consumer's request to switch to an inappropriate offering. However, a Web Service might have to switch a consumer to an offering that is not completely compatible with the currently used one. For example, this can happen after a deactivation of a service offering.

Dynamic Deactivation/Reactivation of Service Offerings
A Web Service can dynamically and automatically (i.e., without a direct request from the consumer) deactivate and reactivate service offerings when changes in operational circumstances affect what offerings it can support. Some service offerings cannot be used in all circumstances. For example, it is sometimes impossible to achieve high QoS or it is dangerous to offer offerings with low security. An example of changed circumstances is unexpected fluctuation in the QoS provided by used Web Services.

The most important issue related to the deactivation of service offerings is what to do with consumers using the deactivated offering. We are developing support for handling such cases. In our solution, the Web Service automatically switches the affected consumers to another offering and then notifies them about the change. When an affected consumer receives such notification, it can decide what to do next in the specific situation. Some examples of consumer decisions are accepting the automatic switching, switching explicitly to another offering that the consumer estimates more appropriate, and discarding the affected operation invocation or the whole session with the Web Service. As consumers in our work are other Web services these decisions are done by some programming logic and not by human intervention. This programming logic can be built into the Web Service or into some external management infrastructure. In cases when there is no appropriate offering to switch to, other approaches to dynamic adaptation, such as finding an alternative Web Service, have to be used. We will discuss later in this paper how our capabilities can still be beneficial in such situations.

The deactivated offering might be reactivated automatically after another change of circumstances and, eventually, the consumers can be automatically switched back to their original offering and notified about the change. This helps achieve as much as possible the originally intended level of service and QoS. A consumer should also have the option of notifying the Web Service that it is not interested in automatic restoration of the original offering. We suggest switching an affected consumer back to the reactivated offering only if the consumer has not explicitly initiated switching offerings in the meantime. Consumers that explicitly initiated switching to an offering (even to the offering provided as the automatic temporary replacement) should only be notified when this offering is reactivated.

Automatic switching after deactivation or reactivation of service offerings helps in fast dynamic adaptation to changes and disturbances, with minimal loss of service and QoS, and with minimal human involvement. It supports both graceful degradation and seamless service upgrades and expansions. In many situations, it is better for an affected consumer to accept another service offering (e.g., with less QoS) from the same Web Service than to break the relationship with the Web Service.

Dynamic Creation of New Service Offerings
We are also working on support for dynamic creation of new service offerings for existing Web services and ports of Web Services. Not all circumstances of run-time operation, such as some issues related to QoS, and not all consumer needs can be predicted in advance. Therefore, this capability is needed as an addition to the concept of service offerings to enable further flexibility, customizability, and adaptability. This capability can also be a useful support for dynamic evolution of Web Services with minimal disruption of heir operation, especially for describing effects on co-operating Web Services. While the former two dynamic adaptation capabilities handle changes that are to some extent anticipated, this capability can be used for unanticipated changes.

Different types of constraints show different need for dy-

namic definition. For example, while many QoS constraints often have to be defined dynamically to correspond to particular operational circumstances, functional constraints change dynamically only when the Web Service is dynamically versioned. For this reason, we relate our research on this capability to our research on the issue of separation and integration of constraint dimensions in WSOL and on the issue of specifying relationships between service offerings.

Note that creation of new service offerings is not creation of new functionality, i.e., Web Service constructs described in WSDL files. It is creation of new sets of constraints (e.g., QoS constraints and access rights) for existing functionality of the Web Service. It can eventually accompany, but not completely replace, dynamic creation of new functionality, e.g., during dynamic evolution of Web Services. Nevertheless, this capability is both powerful and dangerous, in the sense that it cannot be performed arbitrarily due to various possible conflicts. For example, QoS constraints cannot be set arbitrarily because of the limitations of used resources (including other Web Services), mutual dependencies of QoS parameters, and other issues. Detection and resolution of such conflicts can be very complex. Creating new offerings can consume considerable time and resources of the Web Service. For these reasons, we suggest creating new offerings only under certain circumstances, only for certain classes of consumer, and only after rigorous conflict checks. For example, a Web Service might create new offerings when:

- its implementation has dynamically changed (e.g., in the case of dynamic versioning/evolution);
- offerings of a Web Service it uses have been dynamically updated (e.g., offer better QoS);
- an important (e.g., 'premium') consumer has requested creation of a new offering and is willing to pay for it.

In the first two cases, creation of new offerings is initiated by the Web Service itself to better adapt to changed operational circumstances. In the third case, creation of new offerings is done on consumer's demand and can be charged for. One way to limit creation of new offerings to specific classes of consumer is to include appropriate mechanisms only in selected offerings. The infrastructure support necessary for dynamic creation of new service offerings is discussed later in this paper.

Benefits and Limitations of These Capabilities

Compared to finding alternative Web Services and rebinding, our dynamic adaptation capabilities have both advantages and limitations. The main limitation is that service offerings of one Web Service differ only in constraints, which might not be enough for adaptation. Appropriate alternative offerings of the same Web Service cannot always be found or created. In such situations, the consumer has to search for alternative Web Services. However, these capabilities also have their benefits and advantages over finding alternative Web Services and rebinding. These benefits include adaptation speed, enhanced robustness of the relationships between Web Services, simplicity and low overhead, and supplementary support for handling inconsistency in Web Service compositions.

Speed of adaptation is the probably the main advantage. Finding alternative Web Services can take a relatively long time and its success cannot be guaranteed. As Web Services are distributed over the Internet, so are Web Service directories and/or brokers that can be queried to find an alternative Web Service. While some of these Web Service directories and brokers will be federated and synchronized, some will be independent. This complicates and slows the search. Further, even if the process of finding an alternative Web Service is successful, a successful replacement in the service composition is not guaranteed [11, 12]. In modern market and business circumstances, adaptation speed can be an important differentiator among competitors. The suggested dynamic adaptation capabilities do not require human intervention and can be performed very fast. They are particularly faster than finding alternative Web Services when the latter approach requires establishment of new trust relationships.

Another advantage is enhanced robustness of the relationship between a Web Service and its consumer. This improved robustness benefits the Web Service vendor since alternative Web Services can also be provided by the competition. A break in the relationship could mean lost revenues and lost market share. The suggested capabilities help the vendor retain existing consumers and, hence, revenue sources. Additionally, they do not require establishment of new trust relationships. This is an important issue in compositions of e- and m-business Web Services. Due to the issue of trust, the enhanced robustness of the discussed relationship benefits not only vendors of Web Services, but also the consumers.

Further, the suggested capabilities are simpler and incur less overhead than switching between Web Services that provide the same functionality, but with different constraints. For example, switching between service offerings does not require session state transfer and synchronization; the identity of the Web Service does not change so the consumers need not update it; etc. Also, dynamic creation of new Web Services, although possible, is much more complex than dynamic creation of new service offerings.

To conclude, these capabilities are fast, simple, inexpensive (in terms of overhead), and beneficial from the business aspect both on the Web Service and the consumer side. They provide additional flexibility, adaptability, and robustness – qualities that will be very important for choosing between several similar Web Services in the market. We find our approach particularly advantageous when dynamic adaptation is required relatively frequently and can be achieved with a variation, not a drastic modification, of provided services and QoS at the Web Service level (and not completely at the lower system levels). Such circumstances occur in many non-trivial situations, ranging from

small temporary disturbances of service and QoS caused by mobility to dynamic evolution of Web Services.

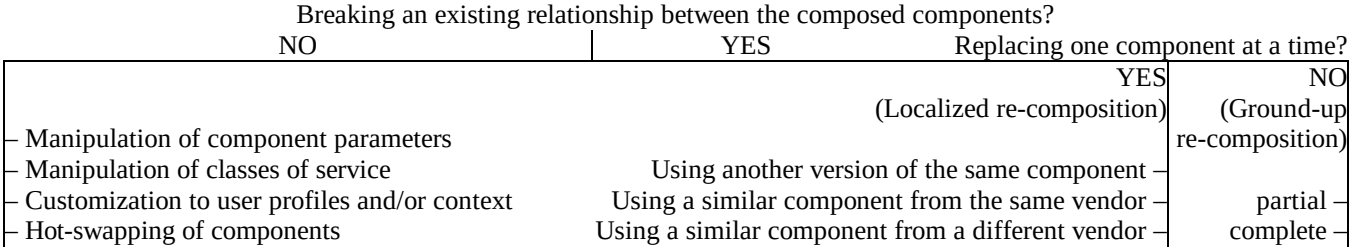

Figure 1. Classification of approaches to dynamic adaptation of component compositions

5 THE DAMSC INFRASTRUCTURE

To support service offerings, the presented dynamic adaptation capabilities, and management activities related to service offerings, we are developing an infrastructure called DAMSC (Dynamically Adaptable and Manageable Service Compositions) and its proof-of-concept prototype. This is still work in progress, so results that are more complete will be presented elsewhere. In the future, we plan to integrate into DAMSC not only support for dynamic adaptation and management using service offerings, but also other approaches to dynamic adaptation of compositions of Web Services. However, our current work is confined to service offerings and it deals more with support to be built into Web Services and less on infrastructure to be developed outside of the composed Web Services.

To enable easy, convenient, and uniform access to dynamic adaptation and management mechanisms we define special management operations. We assume that the signature of these operations and their grouping into appropriate port types will be well known (and hopefully standardized). One of these management operations can provide information about what standardized management port types and/or operations the particular Web Service supports. We believe that as the set of Web Service standards grows to address different aspects of A2A and B2B integrations, it will have to somehow standardize management operations. The signature of our management operations is currently not related to any management standard. If an appropriate management standard for Web Services is developed, we will make our work compatible with it.

We are exploring use of aspect-oriented programming mechanisms and/or SOAP (Simple Object Access Protocol) intermediaries to integrate implementation of these operations, as well as relevant data structures and other support, with already existing Web Services.

One of the management operations supporting service offerings informs a consumer about the currently active offerings it is allowed to use. When a consumer invokes this operation, the operation returns a WSOL file with appropriate service offering descriptions. The consumer might not know at all about the existence of other offerings, which it is not allowed to use, and their features. Another operation informs a consumer about new offerings it may use, again in the form of a WSOL file. This service offering advertisement mechanism can be used when new offerings are dynamically created, when existing offerings are reactivated, when the Web Service vendor allows new classes of consumer to use existing offerings so that it can better utilize resources or to increase its monetary gain, etc. Such advertisement should be done in a way that does not expose security threats to the Web Service. Another set of related operations informs a consumer about the details of the currently used offering. For example, one of these operations returns information about the related service offerings. Note that operations for answering various queries about service offerings, along with those for finding offerings that best match given criteria set, are left for future work.

These operations also support our dynamic adaptation capabilities. In addition, we define several other operations to address issues specific to these capabilities. One operation enables consumers to initiate switching offerings. Another two inform consumers when an offering is deactivated or reactivated. Several operations related to dynamic creation of new offerings are defined. While many consumers will have access to switching offerings, access to operations for deactivation/reactivation and creation of offerings should be restricted.

6 VARIOUS APPROACHES TO DYNAMIC ADAPTATION AND THEIR INTEGRATION

Our dynamic adaptation capabilities are complementary to reconfiguration by rebinding, and are not completely its alternative and replacement. Therefore, it is appropriate to study an integrated approach to dynamic adaptation of component compositions, an approach that would leverage complementary benefits of both manipulations of service offerings and reconfiguration by rebinding. Even further, such an integrated approach could encompass other approaches to dynamic adaptation of component compositions that we have not previously mention in this paper. Let us now try to analyze such approaches and what the characteristics of an integrated approach could be. In the following discussion, we use the umbrella term ‘component’ to emphasize that the discussion applies both to Web Services and software components.

All approaches to dynamic adaptation of component compositions can be classified into two groups, depending whether they break an existing relationship between composed components or not. This is illustrated in Figure 1.

The first group adjusts the relationship between the composed components without breaking it. We identify four sub-approaches and enumerate them in the order of increasing power and complexity. The first is manipulation of some component parameters. The second is providing multiple classes of service and their manipulation. This is the central approach in our current research. The third is re-customization of service to user profiles and/or the operation environment and context. The fourth is hot-swapping the component with a new, more appropriate version (i.e., replacing it during run-time with minimal system interruption). In this sub-approach, the old version of the component is replaced by the new version (maybe even without consumer's knowledge), so there is some preservation of the relationship with the consumer. However, this sub-approach is on the border with the second group, which is discussed next.

The second group of approaches to dynamic adaptation of component compositions breaks up the relationship between the composed components and performs re-composition. This is the reconfiguration by rebinding group. We identify two broad subgroups of approaches. The first one replaces one component at a time – we call this localized re-composition. Note, however, that for a complete adaptation of component compositions often a chain of localized re-compositions is necessary because the replacement of one component with a more appropriate one might introduce inconsistency and the need to replace some of its consumers. The second subgroup breaks larger portions of the composition structure – we call this ground-up re-composition.

In the localized re-composition, we identify three distinct cases, depending how extensive the search for the replacement component has to be. The first case is when instead of the old component, its more appropriate (probably newer) version is used. The difference between this case and hot-swapping of components is that the old version is not removed (i.e., both versions still exist in the overall system), but the consumer switches from using the old version to using the new. The second case is replacing the old component with a similar component from the same component vendor or, eventually, its business partners. The specificity of this case is that it is likely that the component vendor will provide relatively reliable information about the compatibility and differences of its components. This information might be even part of the documentation of the replaced component. Consequently, the search for an alternative replacement component is localized and more straightforward. The third case in this subgroup is replacing the component with a similar component from a different component vendor. This is the most general case and we have briefly compared it with our dynamic adaptation capabilities. To repeat, the main issue is that the search for an appropriate replacement component can turn out to be complex and time-consuming, without guarantees that it will ultimately succeed.

Ground-up re-composition can be complete or partial, depending whether the whole old composition structure is broken or only a part of it. It can be attempted after the chain of localized re-compositions did not succeed or it can be attempted even without trying localized re-composition. We find the latter strategy appropriate when dependencies between the composed components are numerous and/or complex, so that long chains of localized component replacements can be expected.

The approaches to dynamic adaptation of component compositions classified in Figure 1 differ in power, in complexity, and their strengths and weaknesses are often compatible in many aspects. An integrated approach to dynamic adaptation of component compositions would implement several different approaches and apply the appropriate one in a given situation. We believe that although an ideal system for dynamic adaptation would implement the whole spectrum of these approaches, this might not be strictly necessary in practice. At minimum, at least one approach from each of the two big groups (i.e., with and without breaking an existing relationship) should be implemented.

When multiple approaches to dynamic adaptation of component compositions are used, two general strategies can be applied. The first is "start simple, and gradually go to more complexity when needed". In this strategy, a simple adaptation is attempted first. If it produces an acceptable result, then the adaptation process is finished. If the result is not acceptable, the next more complex approach is attempted. The second strategy is "assess the situation and applicability and overhead of different options". First, the given situation is assessed by appropriate program logic. If it seems that simple adaptation is not enough, then the program logic should try to estimate how long the required complex adaptation will last and how this would affect desired system properties such as availability, uninterrupted service, etc. If the duration of the required adaptation process is acceptable, then this adaptation should be attempted. However, if it is estimated that this duration is not acceptable, then the required complex adaptation process should be started only after a temporary good-enough simple adaptation is applied and used in parallel. Let us illustrate this issue with a more concrete discussion.

We advocate using dynamic adaptation capabilities based on manipulation of classes of service of the same Web Service as a complement and addition to finding alternative Web Services and rebinding. Often, the used Web Service will not have a service offering appropriate for dynamic adaptation (and will not be able or willing to create one), so the consumer has to find an alternative Web Service. However, it might be appropriate to automatically switch the consumer to any service offering (of the old Web Service) with at least of some value to the consumer, while searching for an alternative Web Service in parallel. The benefit is that the consumer gets at least some service and QoS while a replacement Web Service is not found. If a replacement Web Service is not found, the consumer has to

make the decision whether to continue using this temporary replacement service offering or to abandon this old Web Service. This example illustrates applicability of our research and its relevance for the theme of this workshop.

7 CONCLUSIONS AND FUTURE WORK

Dynamic reconfiguration of component compositions can further increase flexibility, adaptability, and agility of component-based systems. However, it is a complex issue. It might even turn out to be more complex than dynamic composition, due to system-level requirements, such as uninterrupted service availability, as well as various issues related to paying components. No approach is without drawbacks. For example, reconfiguration by rebinding of Web Services can take a long time, particularly due to the Internet-wide distribution of Web Services and corresponding directories and brokers. Therefore, the strengths of various approaches have to be combined to produce dynamic reconfiguration that can achieve both the required new state of the component system and the desired system-level qualities.

This is where our research on dynamic adaptation of component compositions using manipulation of classes of service fits in. While in this research we study reconfiguration in the context of dynamic adaptation and concentrate on Web Services as specific distributed components, the main ideas, particularly integration of various reconfiguration approaches, seem relevant to dynamic reconfiguration of component systems in general. Our dynamic adaptation capabilities have limitations, but their advantages include speed, enhanced robustness of the relationships between components, simplicity, and low overhead. They can be a useful complement and addition to the reconfiguration by rebinding.

Providing multiple classes of service at the Web Services level also has benefits other than additional support to dynamic adaptation. It gives additional flexibility and choice. Consumers get additional flexibility in selecting appropriate Web Services and their levels of service and QoS, while minimizing the price/performance ratio. On the other hand, providers of Web Services have more flexibility in balancing underlying resources, as well as in covering the Web Service market by addressing the needs of diverse consumers. The overhead related to providing multiple classes of service for Web Services is relatively low, and the complexity of management required is limited.

We continue our work on WSOL by improving specification of the types of constraints supported and by adding some new ones. This work is accompanied by the development of appropriate prototype WSOL tools. However, the main challenges in our future research are related to developing the DAMSC infrastructure and its proof-of-concept prototype.

REFERENCES

1. Aimoto, T., Miyake, S. Overview of DiffServ Technology: Its Mechanisms and Implementation. *IEICE Trans. Inf. & Syst.*, Vol. E83-D, No. 5 (May 2000), IEICE, pp. 957-964.
2. Bosch, J., Szyperski, C., Weck, W. WCOP 2002: Seventh International Workshop on Component-Oriented Programming Call for Papers. *WWW page*, January 31, 2002. On-line at: http://www.research.microsoft.com/%7Ecszypers/events/wcop2002/
3. Curbera, F., Mukhi, N., Weerawarana, S. On the Emergence of a Web Services Component Model. In *Proc. of the WCOP 2001 workshop at ECOOP 2001* (Budapest, Hungary, June 2001). On-line at: http://www.research.microsoft.com/~cszypers/events/WCOP2001/Curbera.pdf
4. The DAML Services Coalition. DAML-S: Semantic Markup for Web Services. *WWW page*, December 12, 2001 (last accessed: March 14, 2002). On-line at: http://www.daml.org/services/daml-s/2001/10/daml-s.html
5. Feng, N., Ao, G., White, T., Pagurek, B. Dynamic Evolution of Network Management Software by Software Hot-Swapping. In *Proc. of the Seventh IFIP/IEEE International Symposium on Integrated Network Management - IM 2001* (Seattle, USA, May 2001), IEEE Publications, pp. 63-76.
6. Hasselmeyer, P. Managing Dynamic Service Dependencies. In *Proc. of the 12th International Workshop on Distributed Systems: Operations & Management - DSOM 2001* (Nancy, France, October 2001). On-line at: http://www.loria.fr/~festor/DSOM2001/proceedings/S5-1.pdf
7. Heineman, G. T., Council, W. T. (eds.) *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, 2001
8. International Business Machines Corporation (IBM), Microsoft Corporation. Web Services Framework. In *Proc. of the W3C Workshop on Web Services – WSWS'01* (San Jose, USA, April 2001), W3C. On-line at: http://www.w3.org/2001/03/WSWS-popa/paper51
9. Kristiansen L. (ed.) Service Architecture, Version 5.0. *TINA-C (Telecommunications Information Networking Architecture Consortium) specification*, June 16, 1997. On-line: http://www.tinac.com/specifications/documents/sa50-main.pdf
10. Magee, J., Kramer, J. Dynamic Structure in Software Architectures. In *Proc. of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering* (San Francisco, USA, 1996), ACM Press, 3-14.
11. Mennie, D., Pagurek, B. An Architecture to Support Dynamic Composition of Service Components. In

Proc. of the 5th International Workshop on Component-Oriented Programming – WCOP 2000 at the 14th European Conference on Object-Oriented Programming - ECOOP 2000 (Sophia Antipolis, France, June 2000). On-line at: http://www.ipd.hk-r.se/bosch/WCOP2000/submissions/mennie.pdf

12. Mennie, D., Pagurek, B. A Runtime Composite Service Creation and Deployment and Its Applications in Internet Security, E-commerce, and Software Provisioning. In *Proc. of the 25th Annual International Computer Software and Applications Conference - COMPSAC 2001* (Chicago, USA, Oct. 2001), IEEE Computer Society Press, pp. 371-376.

13. Oreizy, P., Medvidovic, N., Taylor, R. N. Architecture-Based Software Runtime Evolution. In *Proc. of the International Conference on Software Engineering 1998 - ICSE'98* (Kyoto, Japan, Apr. 1998), ACM Press, pp. 177-186.

14. Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbinger, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., Wolf, A. L. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, Vol. 14, No. 3 (May/June 1999), 54-62.

15. Pryce, N., Dulay, N. Dynamic Architectures and Architectural Styles for Distributed Programs. In *Proc. of 7th IEEE Workshop on Future Trends of Distributed Computing Systems - FTDCS'99* (Cape Town, South Africa, December 1999), IEEE Computer Society Press, 89 –94.

16. Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998

17. Tan, L., Esfandiari, B., Pagurek, B. The SwapBox: A Test Container and a Framework for Hot-swappable JavaBeans. In *Proc. of the 6th International Workshop on Component-Oriented Programming –WCOP 2001* at the 15th European Conference on Object-Oriented Programming - ECOOP 2001 (Budapest, Hungary, June 2001). On-line at: http://www.research.microsoft.com/~cszypers/events/WCOP2001/Esfandiari.doc

18. Tosic, V., Pagurek, B., Esfandiari, B., Patel, K. On the Management of Compositions of Web Services. In *Proc. of the OOWS'01 (Object-Oriented Web Services 2001) workshop at OOPSLA 2001* (Tampa, Florida, USA, Oct. 2001). On-line at: http://www.research.ibm.com/people/b/bth/OOWS2001/tosic.pdf

19. Tosic, V., Patel, K., Pagurek, B. WSOL – Web Service Offerings Language. *Submitted for publication*, 2002.

20. Tosic, V., Esfandiari, B., Pagurek, B., Patel, K. On Requirements for Ontologies for Management of Web Services. *Submitted for publication*, 2002.