Classification of the state-of-the-art dynamic web services composition techniques

Atif Alamri, Mohamad Eid and Abdulmotaleb El Saddik*

Multimedia Communications Research Laboratory – MCRLab
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, K1N 6N5, Canada
E-mail: atif@mcrlab.uottawa.ca
E-mail: eid@mcrlab.uottawa.ca
E-mail: abed@mcrlab.uottawa.ca
*Corresponding author

Abstract: Dynamic web service composition can serve applications or users on an on-demand basis. With dynamic composition, the application's capabilities can be extended at runtime so that theoretically an unlimited number of new services can be created from a limited set of service components, thus making applications no longer restricted to the original set of operations specified and envisioned at design and/or compile time. Moreover, dynamic composition is the only means to adapt the behaviour of running components in highly available applications such as, banking and telecommunication systems where services cannot be brought offline to upgrade or remove obsolete services. In this paper, we present a novel classification of the current state-of-the-art dynamic web services composition techniques with attention to the capabilities and limitations of the underlying approaches. The proposed taxonomy of these techniques is derived based on a comprehensive survey of what has been done so far in dynamic web service composition. Finally, we summarise our findings and present a vision for future research work in this area.

Keywords: web services; composition techniques; dynamic composition; classification; algorithms; measurement; experimentation.

Reference to this paper should be made as follows: Alamri, A., Eid, M. and El Saddik, A. (2006) 'Classification of the state-of-the-art dynamic web services composition techniques', *Int. J. Web and Grid Services*, Vol. 2, No. 2, pp.148–166.

Biographical notes: Atif Alamri received his Master's degree in Information Systems from King Saud University in 2004. He is currently a PhD student at the School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada. His current research interests include dynamic web service composition and web engineering.

Mohamad Eid received his Master's degree in Electrical and Computer Engineering from the American University of Beirut in February 2005. He is currently a PhD student at the School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada. His current research interests include dynamic web service composition and haptic environments and frameworks.

Copyright © 2006 Inderscience Enterprises Ltd.

Dr. Abdulmotaleb El Saddik is an Associate Professor at the School of Information Technology and Engineering (SITE) at the University of Ottawa. He is the Director of the Multimedia Communications Research Laboratory (MCR Lab). He received his MSc (Dipl.-Ing.) and PhD (Dr.-Ing.) degrees in Electrical Engineering and Information Technology from Darmstadt University of Technology, Germany in 1995, and 2001, respectively. His current research focuses on collaborative environments and multimedia communications, web engineering, and haptic audio visual environments.

1 Introduction

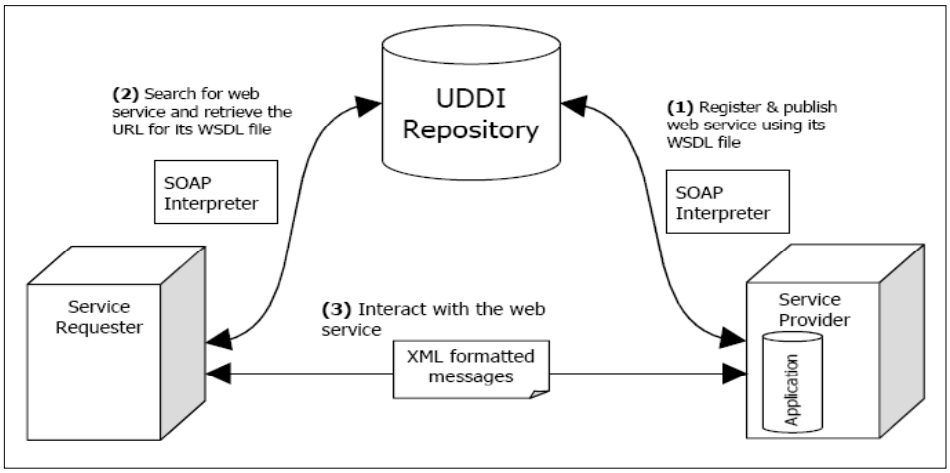
Web services are loosely coupled reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over the internet. Web services can be used alone or in conjunction with other web services to carry out a complex aggregation or a business transaction. A web service is described using a standard that provides all of the details necessary to interact with the service such as, message formats, transport protocols, and location.

Web services adopted the Service-Oriented Architecture (SOA) where different applications can be deployed and accessed by other businesses or services. The communicating applications can be located at a computer or distributed across a network. In fact, web services architecture and standards were developed to be extensible in order to survive for the long run (Booth *et al.*, 2005).

When a service provider wants to publish its web service, it first generates a Web Services Description Language (WSDL) file (Booth and Liu, 2005) to describe its web services with the help of the Simple Object Access Protocol (SOAP) interpreter (Mahmoud, 2005). Then, the service is registered in the Universal Description, Discovery, and Integration (UDDI) repository (Booth *et al.*, 2005) and made available for invocation. The UDDI repository now contains all necessary information to identify this web service along with a URL that points to its corresponding WSDL file. Once a service requestor has queried the UDDI repository and found that this web service best suited its need, it can download the WSDL file of this service and use it to generate messages to interact with the web service by the SOAP interpreter. This flow of events is shown in Figure 1.

Web services architecture enables applications to communicate with other applications. The suitability of web services as a standard framework for the development of conferencing applications in internet telephony has been investigated in El Barachi *et al.* (2005). A case study is presented that includes the definition and implementation of a novel web service for conferencing applications in a SIP environment. Another middleware system named Geo-Located Web Services Architecture (GLWSA) is presented in Linwa and Pierre (2005) to assist mobile clients in discovering geo-located web services and in maintaining the service execution closest to their location context.

Figure 1 Overview of the web services paradigm



Therefore, web services is a key technology that enables business models to move from the Business to Consumer (B2C) model to the Business to Business (B2B) model, which is especially useful in implementing complex business transactions. Moreover, automatic (dynamic or static) composition enables businesses to compile value-added services from elementary services thus forming new services not defined at design times. Also, businesses can discover and bind to interfaces at runtime, consequently minimising the amount of static preparation that is needed by other integration technologies to build customisable applications – not to mention other advantages such as minimising intervention from end users.

In this work, we will survey different techniques used to provide dynamic web services composition and classify them according to their underlying approaches. Web services composition can be categorised into manual or automatic and most commonly static and dynamic. Manual composition means that composition is performed by means of employees who have access to the elementary services, while automatic implies that a software agent performs composition based on some predefined algorithms (Fujii and Suda, 2004).

The composition of web services can be done in a static or dynamic way. It can be done statically by allowing the requestor to build an abstract model of the tasks that should be carried-out during the execution of this web service, at design and/or compile time. This abstract model is nothing more than a representation of a set of tasks and the data dependency among them. However, this should be done before the composition planning starts. Each task contains a query clause that is used to search the real atomic web service to fulfil the task. Static composition is usually implemented through graphs. On the other hand, dynamic composition is achieved by creating the abstract model of tasks and selecting the atomic web services automatically without the interference of the service requestor in the composition process. This type of classification is usually related to the workflow-based composition techniques (Rao and Su, 2004).

In this paper, we will focus more on the classification of dynamic composition techniques rather than the static paradigms. The aim is to reach a better understanding of current dynamic composition techniques and compare the effectiveness of their approaches by highlighting their capabilities and limitations.

This paper is organised as follows. Section 2 highlights the rationale behind dynamic web service composition. Section 3 describes our proposed classification for dynamic composition techniques. For each class, we explain the approach that the class is based upon followed by a literature review of its current implementations and research enhancements. Section 4 summarises the paper by presenting the different capabilities and limitations imposed by each category of dynamic composition techniques. Section 5 presents related work in reviewing web service composition systems and techniques, and shows how our work differs from existing ones. Finally, the last section concludes the paper by summarising our findings and providing directions for future research in the field.

2 Benefits of dynamic web services composition

There are several benefits of the dynamic composition of services. Unlike static composition, where the number of services provided to the end users is limited and the services are specified at design time, dynamic composition can serve applications or users on an on-demand basis. With dynamic composition, an unlimited number of new services can be created from a limited set of service components. Besides, there is no need to keep a local catalogue of available web services in order to create composite web services as is the case with most of the static-based composition techniques.

Moreover, the application is no longer restricted to the original set of operations that were specified and envisioned at the design or compile times. The capabilities of the application can be extended at runtime. Also, the customisation of software based on the individual needs of a user can be made dynamic through the use of dynamic composition without affecting other users on the system (Mennie, 2000).

Dynamic composition infrastructure can be helpful in upgrading an application. Instead of being brought offline and having all services suspended before upgrading; users can continue to interact with the old services while the composition of new services is taking place. This will provide seamless upgrading round-the-clock service capabilities to existing applications (Mennie, 2000).

Tentatively, it should be clear that dynamic composition techniques are much more expensive in terms of computational power and CPU time than static composition, as they impose more complicated algorithms and procedures. Nonetheless, this is not necessarily true all the time, since it is closely related to the application requirements/specifications and the user's needs.

3 Classification of dynamic web services composition approaches

In this section, we introduce our novel classification of dynamic web service composition techniques. We do not claim completeness of survey. We rather classify the techniques into six sections:

1 runtime reconfiguration using wrappers

2 runtime component adaptation

3 composition language

4 workflow-driven composition techniques

5 ontology-driven web service composition

6 declarative composition.

At the end of this section, we present a table that lists a summary of the classes along with their strengths and shortcomings. Also, it is important to mention that some of the reviewed efforts can go underneath more than one class, but we include such systems in the class that we found it to be more tightly coupled with.

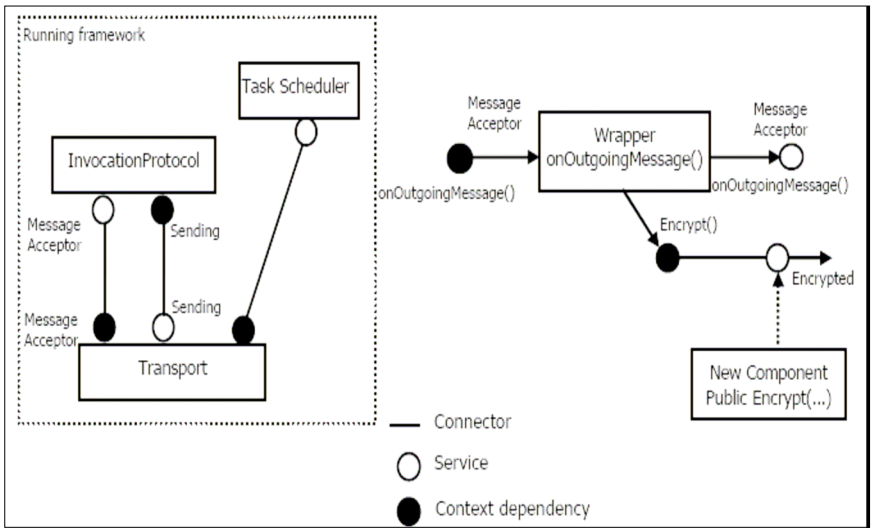
3.1 Runtime reconfiguration using wrappers

The mechanism for introducing new interface behaviour into existing service components is called a wrapper (Truyen *et al.*, 2000). In wrapping, one or more components are wrapped inside another component – called the wrapper – which functions as an interface converter by matching the interfaces of the existing component with a newly introduced one. The wrapper receives a request from other components, does minor changes to make the message interpretable by the wrapped component, and forwards it to the wrapped component. Once a component is identified, a wrapper is used to provide the additional context dependency interfaces to the component so that it can interact with a new component. In this case, a type conflict may be resolved (using common parent types between the component and wrapper). Also, the implementation behaviour of an existing component should be adapted with the new logic introduced by the wrapper.

For example, suppose that an encryption/decryption component must be introduced into the running system of Figure 2, with the intention that some client invocation requests must be sent over the network in a confidential way. To achieve this reconfiguration each remote call has to be encrypted before being sent by a transport component, and at the receiver side, the remote call must be decrypted after its receipt by a peer component. A wrapper is used to provide an additional context dependency interface for encryption/decryption. Also, the wrapper contains logic that implements how the interaction behaviour of the existing component implementation(s) must be extended to corporate with the services of the newly introduced component (Truyen *et al.*, 2000).

The wrapper is controlled by a reconfiguration tool. Before injecting the wrapper, the reconfiguration tool is used to extract the component type of information from a newly introduced component (obtained from service and context dependency interfaces of the component). Based on this information, a new component type manager can be created which corresponds to the new component type. The type manager is then registered with the reconfiguration tool. Now, the wrapper can be inserted into the component. Injection starts from a method in the interface of the type manager that corresponds to the particular implementations to be wrapped. The type manager controls how a wrapper is composed with existing components. It decides when and the order at which the wrapper and the wrapped components should be executed.

Figure 2 Interface adaptation using wrappers



The wrapping approach is a black-box adaptation technique that does not touch the wrapped component's internal implementation. However, it may result in considerable implementation overhead since the complete interface of the wrapped component needs to be handled by the wrapper. This may also lead to excessive amounts of adaptation code and serious performance reductions (Bosch, 1999).

A typical example of a wrapper-based composition system is Proteus (Ghandeharizadeh *et al.*, 2003). Proteus is a system designed to dynamically compose plans that integrate web services, execute the composed plan in the presence of failure and web services migrations, and monitor and show the status of composed components at runtime. It uses techniques to convert online sources into web services and automatically composing XML web services by building wrappers around the service. Wrappers are then converted to XML web service. Network Adaptive Middleware (NAM) is used to minimise the time required to deliver a message to improve the execution of a plan. One advantage of this system is the design of efficient execution of plans by minimising the number of web services invoked and focusing on the efficient transmission of XML files. Also, it uses visualisation tools for the execution of the plan. These tools query the runtime composed components for their status. On the other hand, the system does not support automatic recovery from failure states. This is sufficient during the plan execution when a reference for invoked web services becomes unavailable. Moreover, the composition is based completely on the syntactical representations of web services whose shortcomings are obvious.

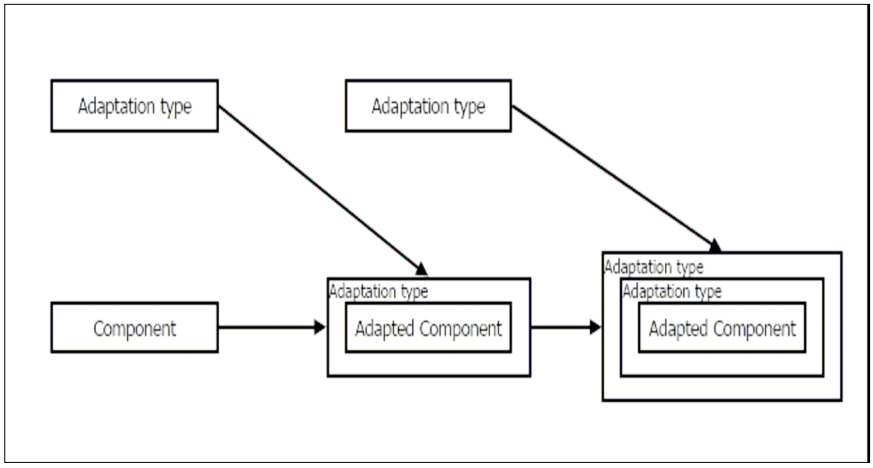
3.2 Runtime service adaptation

This technique involves adapting components into new components or services by changing the interfaces and implementation behaviour of the component at runtime. It is particularly useful for making potentially incompatible components composable. It can be subdivided into two techniques: superimposition and type-safe delegation.

Superimposition (Bosch, 1999) enables programmers to impose predefined, but configurable types of functionality, on the operations a component can perform. By this way, certain behaviour is added to a component in such a way that the complete functionality of the component is affected. Superimposition defines a set of reusable component adaptation types that should be configurable and composable with each other to allow for complex component adaptations.

A graphical representation of superimposition is shown in Figure 3. A basic component is shown that is adapted using two adaptation types. The adapted component is completely enclosed by the adaptations, but neither the clients of the component nor the component itself notices any of these differences (Bosch, 1999).

Figure 3 Adapting a given component using superimposition



On the other hand, type-safe delegation and dynamic component rewiring (Kniesel, 1998) is similar to wrapper-based techniques but it provides typed delegation as a means for components to interact in addition to simple message passing.

In type-safe delegation, a component may have references to other components. Messages to this component, which do not have matching methods, are automatically delegated to the appropriate referenced component. Therefore, some sort of binding operation is required between the referee component and the referenced components (Kniesel, 1998).

What is interesting in this approach is that the referenced component can be reused in an unanticipated way; it introduces no dependencies between refereed and referenced components. Refereed components are adapted automatically to extend the referenced types. Therefore, the resulted refereed component must, in itself, be composable and reusable. Another advantage is that it requires minimum coding efforts (Kniesel, 1998).

In Richards *et al.* (2003), the key goal was to apply and extend an agent factory to use web services as agent components and treat web service composition as a configuration of an artefact. The artefacts have been designed to be redesigned, meaning that they have reusable components. Two levels of agent configurations are supported:

1 conceptual and operational

2 ontology to describe the behaviour, functionality, and the state of the agents and their components.

One limitation of this work is that it defines service templates that must be used to map the conceptual description and the operational description. This means that these templates must be pre-defined. Also, the composed service is not monitored during execution. Besides, it is not clear in the proposed system how the user is going to enter his/her query and the interface used for this purpose. Finally, the user has no control of the composition process.

The work in Mennie and Pagurek (2000) presented an architecture to support composition of service components at runtime to build a single self-contained entity. The choice of 'communication' or 'integration' between composed services is made at the time that a composite service is requested by the user. Composition is made using existing technologies and without the need for a complex compositional language. This work is limited by the fact that it is domain specific because it uses a modified Jini architecture – which is not a web service platform – to create composite services from a network of service components.

3.3 Composition language

A composition language provides a means to define higher-level abstractions that better describe component composition. A composition language is a combination of an Architectural Description Language (ADL), a scripting language, a glue language, and a coordination language.

The ADL is used to specify the component architecture style. The scripting language is used to define various configurations of components for different applications based on the architectural style used. The glue language allows legacy components, that were not designed to be composable with other components, to have plug and play capabilities. Glue can adapt component interfaces, client/server contracts, and platform dependencies. The coordination language is used to specify and configure the coordination mechanisms and policies for concurrent and distributed components (Mennie, 2000).

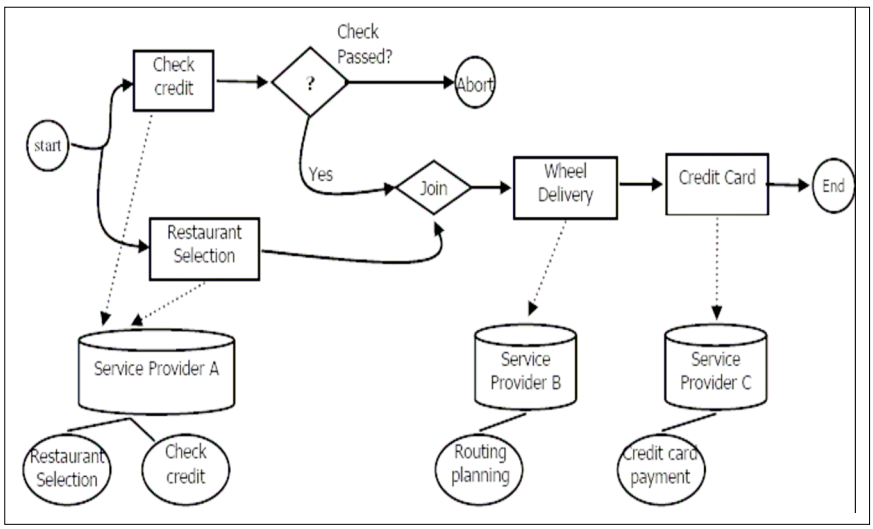
The authors in Narayanan and McIlraith (2002) developed a markup and automated reasoning technique to describe, simulate, compose, test, and verify the compositions of web services. The starting point was the DARBA Agent Markup Language-Services (DAML-S) (Ankolekar *et al.*, 2002) ontology to provide a semantic markup of the content and capabilities of web services. The execution semantic for DAML-S was constructed based on Petri Nets that are bipartite graphs containing places and transitions. The proposed system was implemented and proven to have a broad applicability as a back end to enhance existing manual composition tools or as a stand alone tool for simulating web service composition. In fact, this system does not propose any dynamic service composition technique; rather it proposes automated reasoning tasks for web services compositions. It is more focused on describing the semantics of the composed services using the DAML-S ontology language. Therefore, this work cannot be compared or included in the context of service composition techniques. Maybe it can be used as a software tool for performing reasoning tasks and more oriented towards semantic web applications than dynamic service composition, however, it is presented here as a potential and tightly relevant idea for future investigation.

3.4 Workflow-driven composition techniques

One can argue that a composite service is similar to a workflow where the flow of work items is specified. This technique defines an abstract process model that includes a set of tasks and their data dependencies. Each task contains a query clause that is used to search the real atomic service to fulfil the task (Giacomo and Williams, 2003). This requires the requester to specify several constraints, including the dependency of atomic services, and the user's preference among others. This reduces much of the service composition complications to a constraint satisfaction problem.

In Figure 4, a simple graph describes a composite service that helps customers in organising a ceremony. In the figure, sharp boxes represent invocations of basic or composite services while filled-in circles represent the starting and ending points of the process. Diamond shapes are decision nodes that are used to represent fork and join points, and horizontal bars are used to specify parallel invocation of services and synchronisation after parallel service executions.

Figure 4 An example of workflow composition



For instance, Argos (Ambite *et al.*, 2005) is an architecture for web service composition based on expressive web service descriptions that enable service composition to be automatically derived similarly to the way query plans are generated in a data aggregation system. An ontology for the application domain was defined to integrate data from multiple sources and to provide formal semantics to the sources and operations available (Ambite and Weathers, 2005). The system can automatically generate computational workflows that answer the user query, then it translates the workflow to the XML-based workflow language BPEL4WS for execution. The hierarchical structuring of the ontology provides better scalability of the system. This will significantly reduce the amount of time spent in workflow generation. However, there are no mechanisms for workflow execution monitoring. Moreover, users have almost no control on the composition, thus resulting in an unnecessary execution of compositions that are beyond the user's interests.

3.5 Ontology-driven web service composition techniques

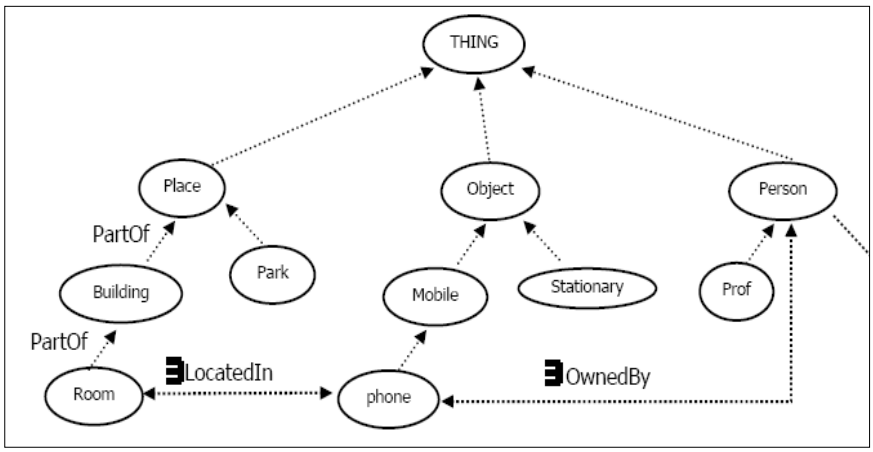
This technique facilitates the semantic dynamic composition of web services. The ontological descriptions and relationships among web services are used to automatically and semi-automatically compose web services. The ontology-driven approaches mainly compose the services based on the goal-oriented inferring and planning (Zang, 2004).

The fact that web ontologies are becoming too large to be used in a single application has stimulated many researchers. For instance, a distributed architecture – called Materialized Ontology View Extractor (MOVE) – is introduced in Bhatt *et al.* (2004a) and elaborated in Bhatt *et al.* (2006) for the optimisation/extraction of a sub-ontology from a large scale base ontology. This work has been extended in Bhatt *et al.* (2004b) to address the issue of semantic correctness of the resulting sub-ontology. Moreover, large distributed ontology framework for tailoring ontologies in the Grid environment has been investigated (Flahive *et al.*, 2004).

Most of the ontology-driven techniques mark-up web service descriptions with ontologies and develop algorithms to match and annotate WSDL files with relevant ontologies. The possible compositions are obtained by checking the semantic similarities between interfaces of individual services (semantic matching) and considering the service quality (QoS matching). Then, these compositions are ranked and presented according to these two dimensions.

A semantic context-based approach for composing web services is proposed in Mrissa *et al.* (2005). The composition is performed based on understanding the semantics of interactions/capabilities of the elementary services. A conceptual architecture that enables ontologies to integrate models, languages, infrastructures, and activities to support reuse and composition of semantic web services is introduced in Pahl (2005).

Figure 5 An example of a domain ontology



To achieve semantic composition, these techniques mostly require a domain-specific ontology design that defines explicit formal specifications of the concepts and relationships among the concepts. It might also require an extraction module that helps us in building ontologies from service profiles. Figure 5 shows a fraction of an ontology. The example shows the taxonomic classification of concepts as well as the relationships that exist between entities.

A dynamic web service composition system, where a service is requested and composed not by its syntax but by its semantic, is proposed in Fujii and Suda (2004). To satisfy the requirement for semantic support, the system comprises three sub-systems: Component Service Model with Semantic (CoSMoS), Component Runtime Environment (CoRE), and Semantic Graph based Service Composition (SeGSeC). CoSMoS integrates the semantic information and functional information into a single semantic graph representation. CoRE provides a unified interface to discover and access components implemented in various component technologies to make them interoperable with CoSMoS components. SeGSeC is a semantic-based service composition mechanism that allows users to request a service using a natural language sentence and it generates the execution path. The major contribution is that this work is semantic-based. Moreover, the CoRE component enables the proposed system to interoperate with other legacy existing systems without any updates to these systems. In addition, the composition technique is somehow controllable by the end users. However, the set of queries that can be used to test the system's functionality is quite limited (not to mention that the system was designed for very limited scenarios and it needs a lot of add-ons to be able to satisfy other types of queries). Finally, the system does not have any monitoring tools for executing the composite services. This implies that execution failures are not monitored at runtime.

Another ontology-based dynamic service composition research is presented in Sirin *et al.* (2003). The authors used the Ontology Web Language (OWL) and DAML-S to provide the semantics needed for web service composition. Notice that DAML-S has been used in a similar manner as for composition language technique (see Section 3.3). However, semantic description is considered as a part of the process for another sophisticated technique. They have developed a service composition prototype that has two basic components: a composer and an inference engine. The inference engine stores the information about known services in its Knowledge Base (KB) and it has the capability to find matching services. The inference engine is an OWL reasoner built on Prolog. Ontological information is written in DAML and is converted to RDF triples and loaded to the KB. The engine has built-in axioms for OWL inferencing rules. These axioms are applied to the facts in the KB to find all relevant entailments. The composer is the user interface that handles the communication between the human operator and the engine. The composer lets the user create a workflow of services by presenting the available choices at each step. The advantage of this work is the use of semantic web for web services composition. Yet, this work suffers from centralisation, thus yielding scalability and availability problems. Moreover, it requires passing redundant messages between the coordinator and other parties, which causes an inefficient use of the bandwidth. Also, it uses filtering on a set of pre-discovered services, rather than dynamic matchmaking and loading.

3.6 Declarative composition techniques

In declarative composition, composite services are generated from a high-level declarative description. The technique uses composability rules to determine whether two services are composable (Dustdar and Schreiner, 2005). Most of the time these rules act as constraints that must be satisfied in order to compose a service. The rules are used to generate composition plans that conform to a service requester's specifications. Techniques that fall under this classification usually tend to reach optimality of

composition against some defined objectives (*i.e.*, cost, time…*etc.*) as they are mathematically modelled. Mostly, the optimality can be achieved by mapping rules to constraints and trying to solve them using operation research methods (Channa *et al.*, 2005).

FUSION is a software infrastructure system that provides the common infrastructure elements needed to support service portals (VanderMeer *et al.*, 2003). Given a user service specification, it automatically generates a correct and optimised execution plan, then executes this plan and verifies the result. The most important advantage of this system is its ability to generate an optimal execution plan automatically from the abstract requirements that a user may specify. Also, this composition system verifies that the result of the execution plan meets the user's requirements, and if not it immediately recovers this execution plan. However, this system creates a bundle of services and uses it to specify an execution plan by choosing services from this bundle. Web services are evolutionarily increasing and determining which subset of these web services to use will limit the choices for the user and will not guarantee the optimality of the result. Also, there will be a probability of using web services that no longer exist.

Thakkar *et al.* (2003) suggests a technique, which extends the 'inverse rules' query reformulation algorithm to generate a universal integration plan to answer a range of user queries. The inverse rules algorithm unites the inverse rules with the user query to produce a datalog program. A technique was developed to map datalog programs to integration plans that can be executed by a streaming, highly parallel execution engine called Theseus. Moreover, a mediator system is described that accepts a user query and returns a URL of a new dynamically composed web service that can answer a class of user queries similar to the user query. The system has many limitations. First, the system does not support any monitoring mechanisms that test the validation of the composition. This may result in improper execution of composite web services. Second, there is no mechanism to automatically model the newly generated services as a data source for upcoming compositions. Third, this technique lacks semantic representation and composition of these services, thus resulting in an error prone composition.

SELF-SERV (Benatallah *et al.*, 2002) is a framework for dynamic and peer-to-peer provisioning of web services. In SELF-SERV, web services are declaratively composed, and the resulting composite services are executed in a decentralised way within a dynamic environment. The framework uses and adapts the state-charts as a visual declarative language. The significant advantage of SELF-SERV is the peer-to-peer service execution model, whereby the responsibility of coordinating the execution of a composite service is distributed across several peer software components called coordinators. Nevertheless, this system does not provide a method to create a composition at runtime for services. Also, it does not consider any semantics of web services during composition decisions. Moreover, this technique imposes some unrealistic requirements that should be implemented by service providers from the partial point-of-view.

Another significant work was performed in Channa *et al.* (2005) to reduce the complexity and time needed to generate and execute a composition and improve its efficiency by selecting the optimal services at the current time. This research proposed an architecture of dynamic web service composition by runtime searching of registries to find services. Therefore, this technique does not use any service template. Moreover, it reduces the dynamic composition of the web services to a constraint satisfaction problem where any linear programming solver can be used to solve it. Also, it insures

the optimality of the web services selection based on domain specific QoS parameters identified by the user. However, this approach does not support user interactive participation in the composition process, which is sometimes important to ensure the user's satisfaction.

4 Putting it all together

Runtime reconfiguration using wrappers does not need knowledge of service implementation because it manipulates the interfaces and message formatting, and adapts the runtime behaviour of the services accordingly. However, the composition is made in a complicated recursive manner, and sometimes, type conflict between passed parameters between services may occur. Runtime service adaptation allows more services or components that are incompatible to be composable with just the necessary functionalities included. Besides, type safe and dynamic delegation is possible. On the other hand, this type of composition requires service codes to be tempered, and as a result, it requires knowledge of the internal implementation of the service. Moreover, naming mismatch may occur in the level of service operations.

Language-criven composition uses standardised interfaces to define a higher level of abstraction which is used to better describe the composition, but in the meantime, services must be designed to be composable (in accordance to the requirement imposed by these standardised interfaces). Workflow-driven composition is user controllable and it requires simple monitoring and recovery. Moreover, recovery plans can be included in the workflow, and distributed execution of a composite service is easier to impose. However, because the workflow plan contains calls to other remote services, the unavailability of service may cause a total failure of execution of the entire composite service, and thus, a composed service cannot be advertised as a new service.

Ontology-driven composition considers semantics in composing services and it supports distributed execution and English-like queries by users. Also, it can be integrated with other types of compositions in order to enhance them. However, performance is an issue for this type of composition because a considerable amount of time is needed to execute the complex logic. Besides, the ontology that is used to fetch for semantics is domain specific and it has no agreed-upon taxonomy. Finally, declarative composition uses mathematically approved equations to derive composition, and hence, can reach optimality of composition. On the other hand, it imposes complex rules and constraints that are hard to implement (sometimes requiring complex solvers), and it is costly in terms of execution time and performance.

As a summary for the six classes of dynamic web service composition techniques explained in the previous sections, Table 1 lists the different classifications with brief descriptions and all the capabilities and limitations we have found for them.

Table 1 Summary of DWSC techniques classification

Classification	*Capabilities*	*Limitations*	*Summary*
Runtime reconfiguration using wrappers	Does not change the code Supports distributed execution Provides additional context interfaces Does not need knowledge of service implementation Adapting behaviour at runtime	Complicated recursive composition Non-optimal code Unanticipated runtime reconfiguration not supported Type conflict may occur	A wrapper is used to make the component capable to interact with a new component.
Runtime service adaptation	Makes incompatible services or components composable New service is produced Necessary functionalities are only included Imposes predefined but configurable types of functionalities Type safe and dynamic delegation is possible	Service code is tempered Requires knowledge of the internal of services No distributed execution Complex monitoring and recovery Naming mismatch No restricted client access mechanisms No class replacement	Adapting components into new services by changing the interfaces and/or behaviour of the component at runtime
Language-driven composition	Specifically designed to assemble components Uses standardised interfaces Defines higher level abstraction to better describe composition	More complicated No recursive composition Components must be designed to be composed	Language specifies the component adaptation and configure coordination and policies.
Workflow-driven composition	Users controllable Simple monitoring and recovery Recovery plans can be included in the workflow. Easier to impose distributed execution of composite services	Composed service cannot be advertised Recursive composition more difficult Unavailability of service causes failure of execution of the composite service.	Builds services to an abstract process and constraints and generate an executable process
Ontology-driven composition	Consider semantics Supports English-like queries Integrated with other technique Distributed composition and execution	Performance. Domain specific No agreed upon taxonomy Assumes semantic service Complex monitoring Failure prone	Ontological descriptions and relationships of web services are used to compose web services.
Declarative composition	Mathematically approved Can reach optimality Does not need knowledge of the internal implementation Allows distributed composition and execution	Uses direct matching Constraints based Uses complex rules Costly in terms of time and performance Requires complex solvers	Using composability rules to check if two services are composable

5 Related work

Despite the amount of research devoted to web services, very little attention has been paid to the understanding of existing dynamic web service composition techniques. We believe that comprehensive comparison and analysis of existing composition technique solutions will result in a better understanding of the current state-of-the-art.

For instance, Rao and Su (2004) present a survey of different approaches in automatic service composition and a comparison study to identify common characteristics and features of an abstract framework. The paper categorises automatic service composition as based on workflow methods or AI planning. It was stated that the workflow approach is most suitable when a process model is defined and where automatic programs are used to find atomic services to fulfil the requirements. On the other hand, the AI planning methods are used when not a process model, but a set of constraints and preferences is defined. One derived conclusion in this work is that the higher the automation does not imply a better composition method: it always depends on the application area. Finally, the authors provide an outlook to essential future research work in the area of service composition.

Moreover, the work in Milanovic and Malek (2004) compares and classifies web service composition – in general – into two major approaches: the industrial approach such as WSDL and BPEL4WS and the semantic web approach including RDF/DAML-S and Golog/Planning. The paper also identifies the requirements for an ideal web service composition approach. Then, it compares current web services composition (static and dynamic) based on the gathered requirements and the problems of modelling, composition, executing, and verifying. An important conclusion from this work is that the short-term composability problem goal should be to adopt an industry standard. A long-term goal must be to incorporate verification mechanisms.

The work in Dustdar and Schreiner (2005) is more related to this work as it presents a literature review of web service composition techniques. Based on some currently existing composition platforms and frameworks, the authors define five categories of service composition:

1 static and dynamic composition (corresponding to design time and runtime composition)

2 model-driven service composition

3 declarative service composition

4 automated and manual composition

5 context-based service discovery and composition.

As a future work, the paper suggests considering non-functional attributes and specifications in the composition process. Unlike the work presented in this paper, our work is specific to dynamic service composition techniques and strategies.

In general, many efforts have been made to survey web services' composition, each of which has several capabilities and advantages. However, to the best of our knowledge, none of these efforts share our literature review to cover various dynamic service composition techniques and systems. We present a novel classification of dynamic composition techniques and accordingly derive the general requirements/specifications that such systems must satisfy/have to perform properly in real time service composition.

6 Conclusion and future work

Web services composition approaches range from those that provide static composition to a more dynamic behaviour of composition. In this paper, we have presented a literature review of current techniques used to compose web services in a dynamic manner. For that, we explored the current approaches that depend on a dynamic behaviour to compose web services, and during which, great attention was made to discuss the capabilities and limitations of each technique. A significant portion of the systems we have described have been implemented and many features are incomplete and many details are yet to be worked out.

From the literature, one can make the following observations:

1 *Semantic composition*

Semantic description, selection, and composition of service components have gained momentum in the last few years. Ontology-driven composition techniques have been envisioned to reduce the complexity and time needed to generate and execute the composition by utilising semantic knowledge. Web service ontology bridges the concept gaps in various parts of the service description and in particular interfaces parameters. Nonetheless, a significant effort must be made to uncover the potential benefits to be gained from applying semantic technologies to other techniques.

2 *Composition languages*

Despite the clear advantages over an object-oriented approach, the composition language approach suffers from significant limitations that make its usage shrinking with time. The main limitation is that many languages need to be compiled and therefore, there is no support for components being plugged in at runtime. The complexity of the approach makes recursive composition a challenge. Moreover, the lack of a standardised composition language questions interoperability among existing implementations. Therefore, an immediate need will be to develop a standard *de facto* language that provides a common understanding and functionalities for composition systems utilising such a technique. As for now, we see diminishing interest among the research community in composition language techniques due to the high complexity and limited interoperability compared to its competitors.

3 *Composition execution*

As far as the execution of composite services is concerned, we envision that the new generation of dynamic web service composition techniques must support centralised, distributed, and/or hybrid paradigms. This trend is motivated by the fact that web service components participating in a composition are distributed in nature. Also, further investigation must be conducted to uncover the benefits and limitations of each paradigm and the scenarios that better suit each paradigm.

4 *Hybrid techniques*

The main drive here is the integration of different techniques in order to improve the overall performance and thus achieve the best of two worlds. The obvious example of such integration is the use of semantic knowledge (ontology) in generating composition workflows, adaptability and wrapping interface matching,

or even in defining the composability rules for the declarative composition. Another possible integration can be achieved by extending the workflow composition technique by modelling the abstract process constraints as composability mathematical rules – from declarative techniques – thus yielding the ability to reach composition optimality.

5 *Runtime reconfiguration using wrappers*

As for the wrapper-based model, the focus is to convert various web sources into web services – the challenge is in running components that were designed without considering such requirements. Yet, there are many concerns: first, multiple wrapping may cause intolerable overhead that overweighs – in some cases – static or semi-automatic composition techniques. Second, ensuring a type-safe wrapping may require semantic knowledge of the wrapper as well as of the existing wrapped component. The latter is more challenging since such knowledge was not envisioned at design time. Therefore, we believe that using ontologies to describe components would facilitate type mismatch-free wrapping. Finally, in some cases, it is sufficient to change the internal implementation of components and go beyond just interface matching.

6 *Service adaptation*

Service adaptation supporters should find ways in which their services can be adapted as needed because they cannot produce abstract software services that satisfy all needs. Therefore, software services or components should be designed to be easily extended and contracted. The insight in to active interfaces is that a component should be flexible enough to handle unforeseen situations.

As a result, we have defined a classification of dynamic web service composition techniques, most of which are still far away from being industrially adopted. Therefore, in our future work we will move forward to develop and/or propose a complete reference model for dynamic web service composition systems based on the requirements and capabilities recognised in this work.

References

Ambite, J.L., Giuliano, G., Gordon, P., Pan, Q., Abbasi, N., Wang, L. and Weathers, M. (2005) 'Argos: dynamic composition of web services for goods movement analysis and planning', *Proceedings of the 2005 National Conference on Digital Government Research*, Atlanta, GA, USA, May.

Ambite, J.L. and Weathers, M. (2005) 'Automatic composition of aggregation workflows for transportation modeling', *Proceedings of the 2005 National Conference on Digital Government Research*, Atlanta, GA, USA, May.

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D. L., Mcllraith, S.A., Narayanan, S., *et al.* (2002) 'DAML-S: semantic markup for web services', *Proceedings of Proceedings International Semantic Web Conference (ISWC)*, Sardinia, Italy, June.

Benatallah, B., Dumas, M., Sheng, Q. and Ngu, A. (2002) 'Declarative composition and peer-to-peer provisioning of dynamic web services', *Proceedings of the International Conference on Data Engineering (ICDE)*, San Jose, CA, March.

Bhatt, M., Flahive, A., Wouters, C., Rahayu, W. and Taniar, D. (2006) 'Move: a distributed framework for materialised ontology view extraction', *Algorithmica, Special Issue on Coarse Grained Parallel Algorithms for Scientific Applications*, to appear.

Bhatt, M., Flahive, A., Wouters, C., Rahayu, J.W., Taniar, D. and Dillon, T.S. (2004a) 'A distributed approach to sub-ontology extraction', *Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA 2004)*, IEEE Computer Society Press, Vol. 1, pp.636–641.

Bhatt, M., Wouters, C., Flahive, A., Rahayu, W. and Taniar, D. (2004b) 'Semantic completeness in sub-ontology extraction using distributed methods', *Computational Science and Applications*, Lecture Notes in Computer Science, Part III, Springer-Verlag, Vol. 3045, pp.508–517.

Booth, D. and Liu, C. (2005) 'Web Services Description Language (WSDL) Version 2.0 Part 0: Primer W3C working draft 3 August 2005', *W3C Technical Reports and Publications*, http://www.w3.org/TR/wsdl20-primer/, (accessed December).

Booth, D., Hass H., Mccabe F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. (2005) 'Web services architecture, W3C Working Group Note 11 February 2004', *W3C Technical Reports and Publications*, http://www.w3.org/TR/ws-arch/, (accessed December).

Bosch, J. (1999) 'Superimposition: a component adaptation technique', *Journal of Information and Software Technology*, April, No. 41, pp.257–273.

Channa, N., Shanping, L., Shaikh, A.W. and Xiangiun, F. (2005) 'Constraint satisfaction in dynamic web service composition', *Proceedings of the Sixteenth International Workshop on Database and Expert Systems Applications*, Copenhagen, Denmark, August.

Dustdar, S. and Schreiner, W. (2005) 'A survey on web services composition', *International Journal of Web and Grid Services*, Vol. 1, No. 1, pp.1–30.

El Barachi, M., Glitho, R. and Dssouli, R. (2005) 'Developing applications for internet telephony: a case study on the use of web services for conferencing in SIP networks', *International Journal of Web Information Systems*, UK: Troubador Publishing, Vol. 1, No. 3, pp.147–159.

Flahive, A., Rahayu, W., Taniar, D. and Apduhan, B. (2004) 'A distributed ontology framework for the grid', *Parallel and Distributed Computing, Applications and Technologies*, Lecture Notes in Computer Science, Springer-Verlag, Vol. 3320, pp.68–71.

Fujii, K. and Suda, T. (2004) 'Dynamic service composition using semantic information', *Proceedings of the Second International Conference on Service Oriented Computing (ICSOC'04)*, New York, USA, November.

Ghandeharizadeh, S., Knoblock, C.A., Papadopoulos, C., Shahabi, C., Alwagait, E., Ambite, J.L., Cai, M., *et al.* (2003) 'Proteus: system for dynamically composing and intelligently executing web services', *Proceedings of the First International Conference on Web Services (ICWS)*, Las Vegas, NV, June.

Giacomo, P. and Williams, S.L. (2003) 'Workflow: a language for composing web services', *Proceeding of Conference on Business Process Management (CBM 2003)*, Berlin, Heidelberg, Germany: Springer-Verlag, October.

Kniesel, G. (1998) 'Type-safe delegation for dynamic component adaptation', *Proceedings of the Workshop on Component-Oriented Programming*, Brussels, Belgium, July.

Linwa, A.C.B. and Pierre, S. (2005) 'A geo-located web services architecture for next generation mobile networks', *International Journal of Web and Grid Services*, Inderscience Publishers, Vol. 1, Nos. 3–4, pp.365–396.

Mahmoud, Q. (2005) *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)*, April, http://java.sun.com/developer/technicalArticles/We-bServices/soa/.

Mennie, D. (2000) 'An architecture to support dynamic composition of service components and its applicability to internet security', *Masters Thesis*, Carleton University, Ottawa, Ontario, Canada, October.

Mennie, D. and Pagurek, B. (2000) 'An architecture to support dynamic composition of service components', *Proceedings of the 5th International Workshop on Component-Oriented Programming (WCOP 2000)*, Sophia Antipolis, France, May.

Milanovic, N. and Malek, M. (2004) 'Current solutions for web service composition', *IEEE Transaction of Internet Computing*, December, Vol. 8, No. 6, pp.51–59.

Mrissa, M., Benslimane, D., Maamar, Z. and Ghedira, C. (2005) 'Towards a semantic- and context-based approach for composing web services', *International Journal of Web and Grid Services*, Inderscience Publishers, Vol. 1, Nos. 3–4, pp.268–286.

Narayanan, S. and McIlraith, S.A. (2002) 'Semantic web services: simulation, verification and automated composition of web services', *Proceedings of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, May.

Pahl, C. (2005) 'A conceptual architecture for semantic web services development and deployment', *International Journal of Web and Grid Services*, Inderscience Publishers, Vol. 1, Nos. 3–4, pp.287–304.

Rao, J. and Su, X. (2004) 'A survey of automated web service composition methods', *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, California, USA: Springer-Verlag, July.

Richards, D., Splubter, S.V., Brazier, F.M.T. and Sabou, M. (2003) 'Composing web services using an agent factory', *Proceedings of AAMAS Workshop on Web Services and Agent-Based Engineering (WSABE)*, Melbourne, Australia, July.

Sirin, E., Hendler, J. and Parsia, B. (2003) 'Semi automatic composition of web services using semantic descriptions', *Proceedings of the ICEIS-2003 Workshop on Web Services: Modeling, Architecture and Infrastructure*, Angers, France, April.

Thakkar, S., Knoblock, C.A. and Ambite, J.L. (2003) 'A view integration approach to dynamic composition of web services', *Proceeding of 2003 ICAPS Workshop on Planning for Web Services*, Trento, Italy, June.

Truyen, E., Joergensen, B.N., Joosen, W. and Verbaeten, P. (2000) 'On interaction refinement in Middleware', *Proceedings of the 5th International Workshop on Component-Oriented Programming*, Cannes, France, June.

VanderMeer, D.E., Datta, A., Dutta, K., Thomas, H.M., Ramamitham, K. and Navathe, S.B. (2003) 'FUSION: a system allowing dynamic web service composition and automatic execution', *Proceedings of IEEE International Conference on Electronic Commerce (CEC 2003)*, Athens, Greece, May.

Zang, R. (2004) 'Ontology-driven web services composition techniques', *MS Thesis*, University of Georgia, Georgia, May.