A reference model for dynamic web service composition systems

Mohamad Eid, Atif Alamri 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: eid@mcrlab.uottawa.ca
E-mail: atif@mcrlab.uottawa.ca
E-mail: abed@mcrlab.uottawa.ca
*Corresponding author

Abstract: Web services have become an emerging and promising technology to design and build complex business applications out of atomic web-based software components. To enforce extensive software reuse and dynamic adaptation, dynamic service composition has experienced an increasing interest in research efforts. Together, the lack of a general conceptual reference model for dynamic web service composition systems and the widespread use of these systems in service-enabled applications constitute a problem of management for these systems. To capture the requirements and challenges of these composition systems, a survey of a representative set of these systems is presented. In this paper, we develop a reference model for describing the functional structure and evaluating the performance of dynamic web service composition systems based on existing dynamic web service composition platforms and prototypes. To the best of our knowledge there has been no such model in the literature.

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

Reference to this paper should be made as follows: Eid, M., Alamri, A. and El Saddik, A. (2008) 'A reference model for dynamic web service composition systems', *Int. J. Web and Grid Services*, Vol. 4, No. 2, pp.149–168.

Biographical notes: 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 technologies and applications.

Atif Alamri received his Master's degree in Information Systems from the 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.

Abdulmotaleb El Saddik University Research Chair and Associate Professor, SITE, University of Ottawa and recipient of the Friedrich Wilhelm-Bessel Research Award from Germany's Alexander von Humboldt Foundation (2007) the Premier's Research Excellence Award (PREA 2004), Canada Foundation for Innovation (CFI) Award (2004) and the National Capital Institute of Telecommunications (NCIT) New Professorship Incentive Award (2004). He is the director of the Multimedia Communications Research Laboratory (MCRLab). He has authored and co-authored three books and more than 160 publications. His research has been selected for the best Paper Award at the 'Virtual Concepts 2006' and 'IEEE COPS 2007'.

1 Introduction

The web services technology is introduced in the following subsections. First, the basic concepts of web services technology such as XML and SOAP are defined and described. Next, we introduce the concept of dynamic web service composition, along with its capabilities and limitations. Finally, a summary of contributions and research goals are outlined.

1.1 Web services

The web services paradigm has emerged as a powerful mechanism for integrating disparate information technology systems leveraging a concept known as Service-Oriented Architecture (SOA).

A web service can be defined as a self contained, language neutral, platform independent, and loosely coupled software component that encapsulates discrete functionality and is described, published, located, and invoked programmatically over standard internet protocols (Booth *et al*., 2005).

Web services are built upon already adopted technology standards namely XML, WSDL (Christensen *et al*., 2005), SOAP (Gudgin *et al*., 2005), and UDDI (Bellwood *et al*., 2005). The web service architecture comprises three major players (Booth *et al*., 2005):

1 The service provider that creates the web service, defines its description, and advertises it to a service registry.

2 The service requester searches the registry and binds to the desired service and invokes it.

3 The service broker/registry that provides a searchable repository of service descriptions.

For more information about web service technologies, architectural models, standards, and applications, the reader is referred to the survey papers (Papazoglou and Dubray, 2004) and (Schlingloff *et al*., 2005).

1.2 Dynamic web service composition: capabilities and limitations

Web service composition involves compiling value-added services from elementary or atomic services to provide functionalities that were not available or defined at design times. It enables quick development of new application functionality through reusing components that collaborate to accomplish a task that cannot be provided by any of the existing services. Traditionally, composition is classified into manual/automatic and static/dynamic. Manual and automatic reflects whether the composition is performed by a human or a software agent. Static composition implies that the compositions is performed at design or compile time. Dynamic service composition, on the other hand, composes an application autonomously when a user queries for an application at runtime. Therefore, dynamic composition involves adapting running applications by changing their functionalities and/or behaviour via the addition or removal of service components at run time. There have been several benefits to dynamic service composition (Mennie and Pagurek, 2000; Dustdar and Schreiner, 2005):

- *Greater flexibility* – 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.
- *New services can be created at runtime* – 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.
- *Users are not interrupted during upgrades of applications* – instead of being brought offline and having all services suspended before upgrading, through the dynamic composition infrastructure, users can continue to interact with the old services while the composition of new services are taking place. This will provide continuous and seamless upgrading service capabilities to existing applications.
- *Unlimited set of services* – unlike static composition, where the number of services provided to 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, theoretically an unlimited number of new services can be created from a limited set of service components.

Despite the clear benefits of dynamic composition, several limitations exist that must be investigated further by the research community. Compared to their static counterparts, the state-of-the-art dynamic composition solutions are complicated and are shown to be slower (Peer, 2005). In addition, significant key additional features must be added to the component model infrastructure to support dynamic binding and the runtime extensibility of components.

1.3 Related work

The idea of developing a reference model for dynamic web service composition systems is, as far as we know, entirely new. Much of the related work has emphasised what has been done so far in web service composition. For instance, Rao and Su (2004) presents a survey of automatic service composition approaches and derives an abstract framework that identifies common characteristics and features. The reviewed

composition approaches are based on either workflow methods or AI planning. 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.

The work in Dustdar and Schreiner (2005) is more related to this work as it presents a literature review and classification of web service composition platforms and frameworks. The authors identify five classes of service composition:

1 static and dynamic composition

2 model driven service composition

3 declarative service composition

4 automated and manual composition

5 context-based service discovery and composition.

Unlike the work presented in this paper, our work is specific to modelling dynamic service composition systems/solutions.

The work presented in this paper is a continuation of our previous work (Alamri *et al*., 2006) where we proposed a novel classification of the state-of-the-art dynamic service composition techniques based on a comprehensive survey of what has been done so far in the field. The techniques were classified 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.

The requirements and reference model proposed in this paper are backed with the discussion regarding the strengths and shortcomings of different classes of the solutions presented in Alamri *et al*. (2006).

1.4 Contributions and goals

The area of dynamic web service composition is very popular today. There exists a confusing set of systems at different development stages and goals. In this paper, our goal is to derive the functional requirements such systems must have in order to be suitable for general dynamic composition needs.

The proposed model may be regarded as an approach towards understanding the functional characteristics of a typical dynamic web service composition system. Because the operational requirements and characteristics are unlikely to be identical across different applications – ranging from business-to-business applications to ubiquitous, mobile environments where available components are dynamic and expected users may vary – the modelling scheme incorporates the basic constituting functional components

commonly used by the reviewed solutions. Furthermore, the proposed model seeks to address the problem of evaluating dynamic service composition systems across different environments.

The rest of the paper is organised as follows. Section 2 provides a literature review of the dynamic composition solutions from which we derive a set of requirements that determine how the reference model should look. Then we propose our model that will act as reference architecture to measure the functional completeness of dynamic composition solutions. In Section 3, we analyse to what extend the available systems qualify against the proposed reference model. Finally, the paper concludes in Section 4 by summarising the paper contents and recommending directions for future work.

2 Literature review and proposed work

In this section, we present an overview of composition systems that have dynamic composition capabilities. Next, we propose the reference architecture that defines and describes the sequence of functions in the composition system as well as the logical flow of data in the composition process. Nonetheless, we have restricted our investigation to a set of prominent and representative systems that are sufficiently mature for reasonable evaluation. Other solutions, such as DySCo (Piccinelli and Mokrushin, 2001) and Semantic Web Service Composition (Mcllrath and Son, 2002), are still in the early development stages.

2.1 Reviewed systems

In this section, we present a brief overview of the existing dynamic composition systems. The reviewed systems are: SeGSeC (Fujii and Suda, 2004), eFlow (Casati *et al*., 2000), Aurora (Marazakis *et al*., 1997), STONE (Minami *et al*., 2003), ICARIS (Mennie and Pagurek, 2000), SELF-SERV (Benatallah *et al*., 2002), Composer (Sirin *et al*., 2002), Ninja (Chandrasekaran *et al*., 2000), SWORD (Ponnekanti and Fox, 2002), SHOP2 (Wu *et al*., 2003), Theseus (Wu *et al*., 2003), Argos (Ambite *et al*., 2005; Ambite and Weathers, 2005), Proteus (Ghandeharizadeh *et al*., 2003), and Fusion (Vandermeer *et al*., 2003):

- *SeGSeC* is an open source semantic-based system designed specifically for dynamic service composition. It allows semantic queries using natural language sentences and it generates execution paths that specify which operations or properties from which components must be executed and in which order. The system also performs semantic verification before executing the composite service.
- *eFlow* is a template-based process model, developed by HP Laboratories, that defines a composite service as a process schema that describes the notion of a generic service node which includes a runtime configurable parameter. The composition is defined as a flow structure (a graph that may include service, decision, or event nodes) that describes the order of execution among the nodes in a process where dynamic adaptation and composition are maintained by dynamically changing the process definition and/or schema.

- *Aurora* is a research project developed at University of Crete, Greece that is based on a container framework that enables dynamic composition of services in open environments by plugging distributed components together into network-centric applications, thus forming a work session. A session manager provides a runtime environment for components, associates appropriate ports of components, handles events, and provides a monitoring infrastructure that tracks the status of sessions.
- *STONE* platform is a distributed system overlaid on the internet that dynamically creates and composes services by combining various functions. It consists of two components: a functional object which represents a component with network connection capability and a service resolver that manages functional objects and composes various services. The composition is represented in a functional executable diagram called a service graph.
- *ICARIS* is a research-based project that provides the required infrastructure to compose services from service components that have been designed for composability (from reputable brokers that have registered in the system). The composition is performed by searching, parsing, and interpreting the service specifications stored in the service components to determine an effective – possibly dynamic - composition technique. The system deploys and provides a caching facility for storing the composite service.
- *SELF-SERV* is a research project system for declaratively composing services within a dynamic environment where the composite service can be executed following a peer-to-peer paradigm. Composite services are specified through state-charts, data conversion rules, and provider selection policies that are translated into XML documents for interoperability.
- *Composer* is a research semi-automatic prototype that guides a user in the dynamic composition of web services by presenting matching services to the user at each composition stage and filtering the possibilities by using a semantic description of the services. To efficiently execute the service, an XML workflow description is generated and passed to the non-centralised system using SOAP.
- *Ninja* is an architecture for composing complex services from simpler services that register themselves within a service discovery service and advertise the structural and semantic information about their interfaces. The platform was developed by Researchers at University of California, Berkeley to enhance availability and fault-tolerance by replicating services and maintaining persistent storage. Moreover, the system defines Ninja paths that describe the sequence of operators and connectors between the source and destination, which when composed performs the necessary service that can be defined at runtime and thus enables dynamic composition.
- In *SWORD*, a research project developed at Stanford University, a service is represented by a rule that expresses its input/output relationship and a rule-based expert system is used to automatically determine whether a desired composite service can be realised or not. The system has a composition and execution model with a customisable filter mechanism that makes the composite service easy to use. The system is designed for automatic static composition, yet can be used for dynamic scenarios.

- *SHOP2* is a domain-independent HTN planning system where a task is decomposed into smaller and smaller subtasks, until primitive tasks are found that can be performed directly. A composite process composition problem is encoded as a SHOP2 planning problem, so that SHOP2 can be used with DAML-S web service descriptions to automatically and dynamically generate a composition of web service calls.
- *Theseus* is a research project that extends the view integration approach to support dynamic integration of data from web services and the dynamic composition of web services. The system uses the inverse rules query reformulation algorithm to generate a universal integration plan to answer a range of user queries. Theseus execution engine queries web services in parallel and streams the data to a join operator that joins information from the services.
- *Argos* is a research project to automatically create computational workflows (compositions) in the presence of aggregation operations. The system defines the application domain ontology, a source description, and the operation programs where all possible workflows that answer a given user query are computed. The resulting operation graph is then translated into an executable XML-based workflow language that can be made as a stand-alone service that can be published and reused by generating the corresponding WSDL specifications.
- *Proteus* is a research project for dynamically composing and executing plans that integrate web services in the presence of failure and web services migrations. The system provides a visualisation tool that queries the runtime components for their status. Efficient execution of the composition plan can be achieved by compressing the XML messages exchanged among components to decrease the execution time of the composite service.
- *Fusion* is a comprehensive software system project for domain-specific service portals. Also, an execution language is developed to describe execution plans in the context of a FUSION service model. A set of data structures and algorithms are developed to generate the correct and optimal execution plan at execution time. The system also has verification and recovery subsystems.

2.2 General requirements

As a result of the conducted literature review and classification of dynamic web service composition techniques presented in Alamri *et al*. (2006), we have specified the main characteristics of a good Dynamic Web Service Composition System (D-WSCS):

- Semantic description and composition of components

  Based on the conducted survey, we have noticed that there is a trend towards a formal semantic representation of service component's interfaces and operations that enable semantic selection and composition of elementary services. Therefore, description and composition using the semantics, in addition to the syntax, minimises the risk of inconsistency among behavioural and interface information (Fujii and Suda, 2004).

- Support of automatic monitoring and recovery mechanisms

  The D-WSCS must be able to monitor and show the status of the composed services at runtime. Also, the use of recovery mechanisms that automatically search for alternatives, in case of composite service execution failure, has shown significant improvement in the performance; even though this requirement becomes quite challenging in distributed environments because of the significant overhead and complexity of tracking the availability of distributed components and their execution status.

- Centralised or distributed execution of service

  It is more likely that service components are hosted by different nodes that make distributed execution an immediate need. Many of the surveyed systems that have a distributed execution capability propose the use of a coordinator that collaborates with the coordinators of other hosts to guarantee a correctly ordered execution of the distributed composite service.

- Context-based and QoS certified composition algorithm

  In addition to type-safe binding of service components, utilising the context information regarding the execution environment increases the composition and execution success rate. Moreover, this information can be used to adjust the composition and/or execution to provide the client with a customised and personalised value-added service. Moreover, checking whether or not the QoS requirements – specified by the user and/or the system – are satisfied by the composition and execution plans has been shown to decrease the execution failure rate.

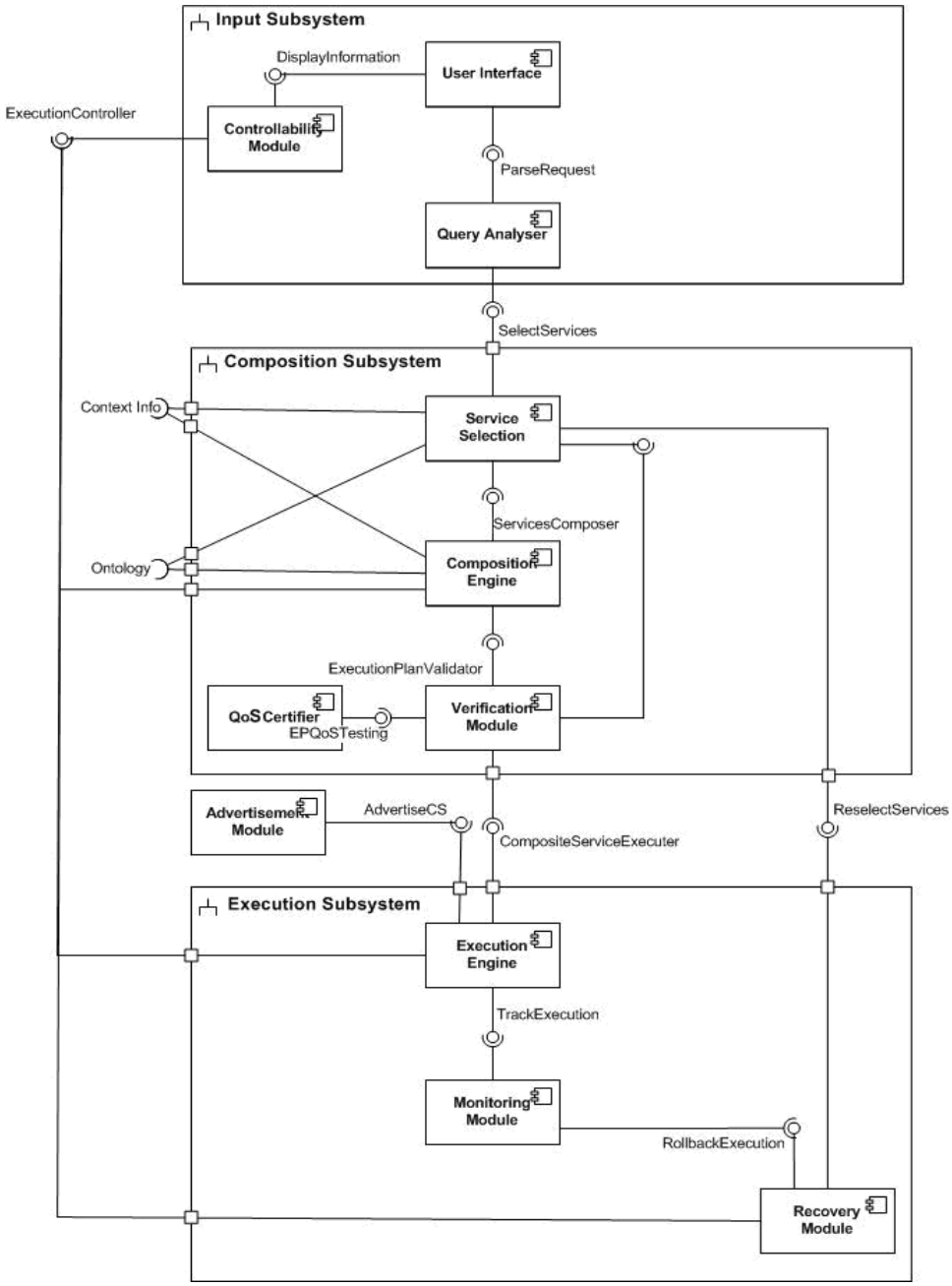
- Controllability

  The D-WSCS must give more controllability to the user on the execution process, and the monitoring and recovery mechanisms. This includes asking the user for composition satisfaction and permission to start execution and selection of the recovery mechanism. At the same time, the system should not go to a limit that will make it too demanding for novice users.

2.3 Proposed reference model

Armed with the general requirements specified in the previous section, we propose our architectural model. We define the dynamic composition system as comprising three subsystems, each consisting of several functional components or modules. These subsystems are: the input subsystem, the composition subsystem, and the execution subsystem. A top view of the proposed reference model is presented in Figure 1.

Figure 1 The reference model for dynamic web service composition systems



2.3.1 The input subsystem

The input subsystem is responsible for receiving the user's request and formulating it in an appropriate form for the use of the composition subsystem. Also, it is responsible for providing the user with suitable interfaces and features for controllability. It is subdivided into the following components:

- *The User Interface* – This component is responsible for handling the interaction between the user and the whole system. It identifies the basic communication channel with the user. It can be a simple graphical user interface which contains GUI components that the user can use to query and control the composition and see the monitoring and result information.
- *Query Analyser* – Basically, the Query Analyser receives the user request from the GUI components and parses them into a concrete set of requirements. The user query format differs significantly from one system to another, ranging from natural language sentences to logical formulas, in addition to being defined as a pair of inputs and outputs. After parsing the request, the Query Analyser formulates the requirements into a form that is interpretable by the Service Selection Module. Finally, the Query Analyser passes the formatted requirements to the Service Selection Module by calling the proper method in its interface.
- *Controllability Agent* – This agent provides users with controllability in the composition process, execution process, the recovery mechanisms, and in some cases, the selection of elementary services. Mainly, the user is prompted for execution permission after the composite service is generated. Moreover, this component is responsible for passing the final result of the composite service execution as well as the monitoring and recovery information.

2.3.2 The composition subsystem

This subsystem is responsible for selecting and composing elementary, and possibly composite, services to provide value-added behaviour based on the user's specific requirements. To do so, this subsystem is functionally subdivided into the following components:

- *Service Selection Module* – Logically, the Service Selection Module is the first component of the composition subsystem. This module is responsible for selecting web services that best satisfy the user's formulated requirements that were provided by the Query Analyser. The selection mechanism correlates one or more types of service descriptions/user information (such as semantic, syntactic, and/or context) at runtime to reach the best combination of web services that suits the user's requirements. The functional and/or non-functional QoS parameters come into play here as the selection criteria to select qualified services for composition.
- *Composition Engine* – The composition engine defines the procedures of binding or coordinating the group work of the selected services to provide more complicated services. It answers the question of how to compose the services. The output of this component can be an integration plan, a workflow, or a reference to a composite web service. The composition engine may use ontologies and context information.

- *Verification Module* – It verifies that the composite service is error free and satisfies the user's QoS requirements. The verification module queries the QoS Certifier for non-functional QoS parameters satisfaction. When the composite service fails to satisfy the user QoS requirements then the verification module returns the control back to the service selection module indicating a satisfaction failure problem to select another composition. Otherwise, the verification module forwards the composite service to the Execution Engine.
- *QoS Certifier* – This component checks whether the composed service satisfies the non functional requirements specified by either the user and/or the system. This is due to factors like execution time, available bandwidth, response time, and authenticity that play a prominent role in making execution decisions (*e.g.*, services that require a third party validation, a banking processing request or those that run on limited pervasive devices).
- *Context Information Source* – The Context Information is utilised by the web service composition system to adjust selection, composition, execution, and/or output format to provide the client with a customised and personalised behaviour. Context information can be a customer's name, address, location, type of device, and preferences of communication.

2.3.3 The execution subsystem

This subsystem is responsible for executing the composite service and monitoring the status of the components at runtime. Also, in case of failure, the execution subsystem is responsible for recovering from the failure state to the original consistent state and bringing such information to the user's attention. Functionally, we divide the execution subsystem into the following three components:

1 *Execution Engine* – It executes the composite service by invoking individual components and passing messages between coordinating components. Execution can be classified as centralised, distributed, and hybrid. Centralised execution is similar to the client/server paradigm in which the server acts as a central scheduler that controls the execution of the participating components. In contrast, in distributed execution, each service residing at a peer in the network has a coordinator that collaborates with coordinators of other services residing on different hosts to guarantee a correct ordered execution of the composite service. Hybrid execution is also an option where the coordinator manages more than one service. Afterward, the execution engine sends the result of the execution back to the user interface module through the controllability agent to confirm execution of the composite service.

2 *Monitoring Module* – During the execution of a composite service, errors or failures may happen (examples are unavailability or type mismatch between composing component's interfaces). Therefore, the monitoring module is responsible for monitoring and showing the status of the composed services at runtime. Also, it is responsible for invoking the recovery module in case a failure was detected.

3 *Recovery Module* – The monitoring module forwards a failed composite service to the recovery module, which supports automatic recovery from failure states. This involves looping back to the composition or selection components to recompose the services or to select an alternative component in case a component is no longer available or has been moved. Moreover, the recovery module handles exceptional states during the execution of the components without the service being aborted.

3 Analysis of existing systems

In this section, we analyse to what extent the available systems are functionally comprehensive. For this purpose, we evaluate these composition solutions with our requirements for dynamic composition techniques outlined earlier in the previous section. Accordingly, we examine the fulfilment of the requirements concerning the input subsystem, the composition subsystem, and the execution subsystem. We conclude this section with a summary of our findings. For a better orientation in the ensuing discussions, we have visualised the results of our analysis in Table 1. The table benchmarks the reviewed systems against eleven features/components (query analyser, semantic source, control agent, *etc*.) to measure to which extent does each system support the corresponding functionalities. In the following sections, we advocate the conclusions/summaries presented in Table 1.

3.1 Input subsystem

3.1.1 Query Analyser

Many existing systems – namely eFlow, Aurora, STONE, ICARIS, SELF-SERV, Ninja, SWORD, SHOP2, and Proteus – do not define how a user's query for a composite service is generated and how this request is translated into a concrete set of requirements. These systems assume that a user request is given as a pair of inputs and outputs or as a logical formula.

SeGSeC, Argos, and FUSION specify the format of the user request. In SeGSeC, a RequestAnalyser parses a user request given as a string text into a CoSMoS semantic graph representation to be used by the ServiceComposer component. Argos uses a restricted ontology approach for analysing English-like requests using concepts that are present in actual sources and/or are data in expected user requests. FUSION defines User Specification Subsystem (USS), a graphical form-based interface that allows the user to specify abstract requirements and convert them into more structured forms that are consumable by the plan generator.

Composer and Theseus provide partial support for input subsystems. Composer asks the user to choose which components it should select during its service composition process. Therefore, understanding the structure of the requesting service may be too demanding for novice users. Theseus requires that a user request follows a specific syntax and by using a mediator system, it can be reformulated into a datalog program representing a set of source queries.

Table 1 Analysis results

System	*Query Analyser*	*Dynamic Selection*	*Composition Template*	*Verification Model*	*Distributed Execution*	*Monitoring Module*	*Recovery Module*	*QoS Certifier*	*Control Agent*	*Semantic Based*	*Context Source*
SeGSeC	■	■	—	■	—	—	—	—	■	■	□
			Execution plan	Centralised							
eFlow	—	—	■	□	■	■	■	□	—	□	—
				Centralised							
Aurora	—	■	■	—	■	■	■	—	—	—	—
				Distributed							
STONE	—	■	■	□	■	—	—	—	—	—	□
				Distributed							
ICARIS	—	—	■	—	■	■	□	—	—	—	—
				Distributed							
SELF-SERV	—	■	■	—	■	—	—	—	—	—	—
				Distributed							
Composer	□	□	■	□	■	—	—	—	□	■	■
		Semi-automatic		Centralised							
Ninja	—	■	—	—	■	■	■	—	—	—	—
			Ninja Paths	Distributed							
SWORD	—	—	—	■	—	—	—	—	—	□	—
			Compos. plan	Centralised							
SHOP2	—	■	—	—	—	■	□	—	—	■	■
			SHOP2 Planer	Centralised							
Theseus	□	■	—	—	■	—	□	—	■	—	—
		Filtering	Integration Plan	Centralised							
Argos	■	■	—	—	■	—	—	—	■	■	■
			Workflow	Centralised							
Proteus	—	■	—	—	■	■	■	□	—	—	—
			Integration Plan	Centralised							
Fusion	■	■	—	■	—	■	■	—	□	—	—
			Execution Plan	Centralised							

Notes: ■ support, □ partial support, — no support.

3.1.2 Controllability agent

Controllability involves the ability of a system to provide execution and monitoring/recovery supervision by the users. This seems contradicting with the fundamental composition requirement; we mean automatic and dynamic composition. Therefore, none of the reviewed systems provide controllability over the execution process. Nonetheless, some claim semi-automatic composition by asking the user's for permission to execute a composed service, and to show alternatives if the user disagrees with the composed one. This approach is supported by SeGSeC, Theseus, Argos, and partially by Composer. In addition, Fusion allows users to specify a set of methods to be invoked and a logical expression that represents the user's satisfaction conditions, thus giving significant composition planning to the end users.

3.1.3 Support of semantics

Service composition could be semantic-based so that a service and its interfaces are selected and composed not by their syntax but rather by their semantics. Users can specify their needs semantically without explicitly referring to a low level syntactical description of services. Many existing systems – namely Aurora, STONE, ICARIS, SELF-SERV, Ninja, Theseus, Proteus, and FUSION – neither specify how to represent or model the semantics of components nor do they support semantic composition. For example, Ninja only compares and matches the data types of interfaces but does not consider any semantic information at all. This reduces the flexibility of finding and correlating semantically similar services significantly.

To achieve the ambitious goal of semantic-based dynamic service composition, both the modelling of components and the composition technique must support semantics. SeGSeC, Composer, SHOP2, and Argos propose the use of domain-specific ontologies to describe concepts and relationships against which components are examined for composition. For instance, SeGSeC defines a semantic graph that integrates the functional and logical aspects of a component. A reasoner checks whether the semantics of the execution graph satisfies the user's request. The reasoner derives goal statements from the user's query and facts from the execution path and then checks if the goals can be derived from the facts by applying predefined rules. Finally, it is worth mentioning that SWORD and eFlow partially support semantic services as they require the post/preconditions and component interfaces that can be defined independently using different annotations, which may result in inconsistencies.

Nevertheless, there are considerable limitations. On the one hand, the ontology definition is domain limited and its effectiveness is not measurable. Therefore across-domain dynamic service composition is not semantically supported yet. On the other hand, defining an agreed-upon taxonomy as a foundation of ontology is a big challenge. These substantial limitations seriously compromise the ontology-based dynamic service composition techniques' eligibility for standardisation.

3.1.4 Context source

Context is the information that characterises the interaction among entities and their surrounding environment (Dey *et al*., 2001). The composition process will be subject to personalisation in order to meet the user's preferences. Most of the surveyed systems do not support context-aware composition. The only exceptions are Composer, SHOP2,

and Argos where they integrate the context and semantic information during a service composition. Parameters that are considered as context information are those such as the execution environment, which includes device and network capabilities, user preferences, address and current location, among others.

3.2 Composition subsystem

The composition subsystem has three functional requirements: the selection of services for composition, binding the services using one of the previously reviewed techniques, and a verification mechanism to check for the validation of the composite service before execution.

3.2.1 Selection module

All systems provide selection mechanisms based on a concrete set of requirements provided by the input subsystem. However, Composer selects the service in a semi-automatic fashion by presenting matching services at each step of composition. To limit the number of services that are possible matches for the composition requirements, filtration is introduced in both Composer and Theseus that is based on the profile description of services.

3.2.2 Composition engine

The composition engine has the task of providing complex behaviour from a set of atomic services whose behaviours are predicted. It defines composition strategies that are based on predefined templates, execution plans, workflows, or integration plans. eFlow, Aurora, STONE, ICARIS, SELF-SERV, and Composer are dynamic composition systems which define abstract templates and select components to fill in the template at runtime. Those template-based systems, however, are unable to compose a service unless a template already exist that implements the requested service. The template-based systems are suitable for B2B applications where interactions are complex but static, but less suitable for pervasive applications where applications and users requirements are more likely to evolve in real time.

SeGSeC, SWORD, and FUSION do not require any template to compose a service. Instead, they generate an execution path by computing all possible combinations of the components. Then the execution path is extended until all the requirements are met and the operations in the execution path become executable. That is, all the inputs of the operations either are specified in the user's request or are provided as the properties of the components. Theseus and Proteus concentrate on generating integration plans to answer user queries by correlating information from different web services. Argos defines composition as a workflow that is a set of services linked together with the control and data flow among the services. Ninja has introduced a very similar concept to workflow named Ninja path, which defines an ordered sequence of compatible services and data streams to make services interoperable. SHOP2 defines a process model that includes a designation of a top-level composite process into a structured collection of sub-processes. In this case, the output of a composition is a composite process.

3.2.3 Verification module

It is quite common that the composite service must be evaluated before being executed. SeGSeC, SWORD, and FUSION support reasoner to check whether the composition meets the user's request. Aurora, ICARIS, SELF-SERV, Ninja, SHOP2, Theseus, Argos, and Proteus do not support any verification mechanisms. STONE and Composer provide partial support for verification. The STONE platform has a major component called service resolver that handles the verification issue.

3.2.4 QoS Certifier

While all the investigated solutions address different functional QoS aspects (such as scalability, capacity, reliability, stability, among others), in fact, only a few solutions touch upon the issue of non-functional QoS aspects such as execution time, throughput, availability, and security. As a matter of fact, eFlow supports some QoS features like events and exceptions handling, ACID service-level transaction, monitoring and security management. Moreover, Proteus focuses on efficient execution of composition plans by reducing the number and size of exchanged messages among collaborating components in a composition. The messages are compressed to minimise the time required to send messages, which in turn minimises the overall execution time.

3.3 Execution subsystem

3.3.1 Execution engine

The execution subsystem performs the composition plan by accessing the components, invoking the operations, and retrieving properties of components in a specific order. Basically, two approaches for composite service execution can be distinguished: centralised and distributed execution. Centralised execution is similar to the client/server paradigm in the sense that a central scheduler controls the execution of the components in the composite service. The eFlow platform, SeGSeC, Composer, SWORD, SHOP2, Theseus, Argos, Proteus, and FUSION work with a centralised scheduler. The obvious advantages are simplicity and manageability, while the main drawback is the single point of failure issue that is inherited from the client/server paradigm.

The distributed paradigm, in contrast, expects the participating components to share their execution context in a peer-to-peer fashion. Every host is assumed to have a coordinator, which has to collaborate with other coordinators to guarantee a successful execution of composition plans. Aurora, STONE, ICARIS, SELF-SERV, and Ninja use such a distributed execution system. It is worth mentioning that the capabilities of some systems (examples are Aurora and SELF-SERV) can be extended to support hybrid execution by enabling the coordinator to control not only one but a set of services on the single host.

3.3.2 Monitoring module

Execution monitoring is concerned with notifying or invoking other components when certain events at the monitored component occur, in addition to tracking the progress and the current state of service flows as well as maintaining the interaction history for each participant.

Out of the investigated solutions, eFlow, Aurora, ICARIS, Ninja, and SHOP2 are the ones providing the most support for execution monitoring. Aurora supports the distributed monitoring infrastructure for logging and correlating event records (important to gain a global understanding of the interaction history of a service flow). Ninja tackles monitoring by looking for transmission errors at the stream level so that every IO operation is checked for failure and the appropriate action will take place accordingly.

On the other hand, SeGSeC, STONE, SELF-SERV, Composer, SWORD, Theseus, and Argos do not employ any monitoring functionalities in their composition processes. On one hand, this will significantly increases the rate of execution failure among composite services, and it reduces the controllability of execution on the other hand.

3.3.3 Recovery module

The recovery module is usually tightly coupled with the monitoring module as it represents the reaction of detecting execution errors or failures. The only exceptions to this are ICARIS and Proteus in which full monitoring capabilities are supported while recovery is partially supported. For instance, Proteus provides visualisation tools that bring the failure data to the attention of the user and s/he manually recovers the execution by selecting other components or by aborting the actions. ICARIS specifies that if a conflict is detected by the monitoring infrastructure, the composition is aborted. This will not suffice since even if the structural composition was successful, the behavioural (functional) composition may not be successful.

3.4 Summary

As a summary, we can say that none of the investigated solutions suffice for all the requirements of the general-purpose dynamic web service composition system. Even though, if we take a look at Table 1, there are systems like eFlow, SeGSeC, and Fusion which cover quite a lot of these requirements, this should not belie the substantial limitations that seriously compromise their eligibility for dynamic web service composition (such as providing monitoring and recovery mechanisms, in addition to semantic or context composition sources). Notice that most of the analysed systems are ongoing researches and thus the analysis results are based on the current states of these projects at the time this document was written.

4 Conclusion

In this paper, we have investigated various solutions for dynamic web service composition systems. Starting with a detailed introduction of dynamic web service composition fundamentals and solutions followed by an observation of the key characteristics of these solutions, we have derived an extensive set of functional requirements that should be supported by dynamic composition systems. In regard to these requirements, we have thoroughly analysed a variety of current state-of-the-art dynamic composition systems – commercial systems, open source projects, as well as research efforts.

Despite the clear benefits of the proposed functional architecture, a few shortcomings should be highlighted. First, even though the components are loosely coupled, there exist logical dependencies among their implementations. Second, the interfaces of these components are not defined as they depend on the implementation of each component. Finally, the proposed model is not completely general due to the diversity of functionalities/capabilities of the reviewed systems. For instance, service selection is not always before the composition engine. In fact, some systems propose that the composition plan is created before selecting the participating services.

With our analysis of existing dynamic composition solutions, we have revealed dissimilarities in the reviewed systems. The focus of our criticism is that the solutions are largely diverging in their functionalities and capabilities to the extent that some of these functionalities become incomparable. For instance, systems – like SeGSeC and Composer – were designed to accommodate semantic composition of services while others – such as Proteus and Fusion – do not consider any semantic information in the composition processes. This has a considerable number of negative consequences regarding the interoperability of these systems. Therefore, our proposed model may provide the 'guide lines' for dynamic web service composition systems developers.

For the dynamic composition process, we see the need for a new generation of dynamic composition systems, which recognise the central importance of exploiting the semantic description of web services to facilitate the dynamic composition of web services. It seems that the necessity of using semantic definitions to achieve satisfactory composition is becoming more and more recognised. At the same time, these solutions should not neglect other important issues related to context-aware composition and execution of the resulting composite service, and the classic composition functionalities such as automatic service discovery and a reliable means for monitoring and recovery.

As for the execution component, we see that supporting distributed as well as centralised executions of a composite service is becoming a necessary and sufficient functionality because elementary services are distributed in nature. This requirement empowers the need for a coordination component that participates in the network working on behalf of the system to ensure a logically ordered execution of the composite service components.

References

Alamri, A., Eid, M. and El Saddik, A. (2006) 'Classification of the state-of-the-art dynamic web services composition techniques', *International Journal of Web and Grid Services*, Vol. 2, No. 2, pp.148–166.

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: Westin Buckhead.

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: Westin Buckhead.

Bellwood, T., *et al*. (2005) 'Universal Description, Discovery and Integration specification (UDDI) 3.0', *W3C Technical Reports and Publications*, http://uddi.org/pubs/uddi-v3.00-published -20020719.htm (accessed December).

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

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).

Casati, F., Ilnicki, S., Jin, L.J., Krishnamoorthy, V. and Shan, M.C. (2000) 'Adaptive and dynamic service composition in eFlow', *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, Stockholm, Sweden.

Chandrasekaran, S., Madden, S. and Ionescu, M. (2000) 'Ninja paths: an architecture for composing services over wide area networks', *CS262 Class Project Write-up*, UC, Berkeley.

Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. (2005) 'Web Services Description Language (WSDL) 1.2', *W3C Technical Reports and Publications*, http://www.w3.org/TR/wsdl/ (accessed December).

Dey, A.K., Salber, D. and Abowd, G.D. (2001) 'A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications', *The Human-Computer Interaction (HCI) Journal*, Vol. 16, Nos. 2–4, pp.97–166.

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

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.

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, Nevada.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J. and Nielsen, H.F. (2005) 'Simple Object Access Protocol (SOAP) 1.1', *W3C Technical Reports and Publications*, http://www.w3.org/TR/SOAP/, 2001 (accessed December).

Marazakis, M., Papadakis, D. and Nikolaou, C. (1997) 'The Aurora architecture for developing network-centric applications by dynamic composition of services', *Technical Report TR 213*, FORTH/ICS.

Mcllrath, S. and Son, T. (2002) 'Adapting Golog for composition of semantic web services', *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, Toulouse, France.

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.

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.

Minami, M., Morikawa, H. and Aoyama, T. (2003) 'The design and evaluation of an interface-based naming system for supporting service synthesis in ubiquitous computing environment', *Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J86-B, No. 5, pp.777–789.

Papazoglou, P. and Dubray, J. (2004) 'A survey of web service technologies', Technical Report DIT-04-058, Department of information and Communication Technologies, University of Trento.

Peer, J. (2005) 'Web service composition as AI planning – a survey', Technical report, University of St. Gallen, Switzerland.

Piccinelli, G. and Mokrushin, L. (2001) 'Dynamic e-service composition in DySCo', *21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*.

Ponnekanti, S.R. and Fox, A. (2002) 'SWORD: a developer toolkit for web service composition', *Proceedings of the Eleventh World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii.

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.

Schlingloff, B.H., Martens, A. and Schmidt, K. (2005) 'Modeling and model checking web services', *Electronic Notes in Theoretical Computer Science: Issue on Logic and Communication in Multi-Agent Systems*, Vol. 126, No. 3.

Sirin, E., Hendler, J. and Parsia, B. (2002) 'Semi-automatic composition of web services using semantic descriptions, workshop on web services: modeling, architecture and infrastructure', *Conjunction with International Conference on Enterprise Information Systems (ICEIS)*, Ciudad Real, Spain.

Vandermeer, D.E., Datta, A., Dutta, K., Thomas, H.M., Ramamritham, 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.

Wu, D., Parsia, B., Sirin, E., Hendler, J. and Nau, D. (2003) 'Automating DAML-S web services composition using SHOP2', *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, USA.