INCREMENTAL WEAK COMPOSITION AND INVOCATION OF GEOGRAPHIC WEB SERVICES

Carlos GRANELL, José POVEDA, and Michael GOULD
Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I
E-12071, Castellón, Spain
{canut, albalade, gould}@uji.es

ABSTRACT

Geographic web services will soon become subsumed in the e-commerce world, a world where the composition of simple or atomic services to build compound services is a key characteristic. Since currently no detailed model of composite geographic web services exists, and within the scope of work on a European Union-funded project, we define such basic composition as part of an effort to test for geographic web service interoperability. Rather than adopting current static methods, we build on the concept of weak composition and provide a model for defining, composing and invoking compositions in a flexible manner. We demonstrate a prototype application for this purpose, and illustrate its utility through a simple arithmetic function composer scenario in which complex arithmetic expressions could be easily evaluated in base of the composition of atomic services. Benefits of this weak composition model over process-oriented alternatives are mentioned. Extension to geographic web services within an emergency management use case is proposed and necessary semantic and other extensions are outlined.

KEY WORDS

Web services composition, weak composition, semantic interoperability, interoperability testing.

1. INTRODUCTION

E-commerce has become so important over the past few years that it promises to become the global business paradigm of excellence in the near future. E-commerce services will naturally subsume the niche market of geographic information services, which are the ultimate focus of the work reported here. One of the principle characteristic for a mature and consolidated e-commerce will be to offer not only atomic services to be used ad hoc but also compound services based on the chaining of other services (atomic or also compound) which are then invocated using web technologies. The study of workflow (and corresponding services) has over the past few years provided techniques for modelling the flow of activities, normally known *a priori*, that are executed procedurally within a controlled, homogeneous intra-organizational environment, and where a single participant controls the flow. Within the web context, however, it is not always possible to encounter services which are homogeneous with respect to their composition. In the heterogeneous and dynamic context that is the web, diverse services may appear and disappear without warning. For electronic commerce to reach its expectations, it would seem unwise to attempt migration of traditional process modelling techniques to these new open environments [1]. It would seem more logical to adapt the process models to inherent positive characteristics of the web, which implies weak composition (akin to late binding). For this reason, and within the realm of the European Union-funded project *ACE-GIS* [2] we propose a novel approximation defining a declarative composition model which is consistent with the open and dynamic characteristics of web architecture and behaviour. *ACE-GIS* proposes to build a developers platform for model-driven design and invocation of compound geographic information services (in addition to basic e-commerce services such as authentication and eventually payment). While judging the conformance of a novel web service to the implementation specifications within Open GIS Consortium is quite straightforward[13], the leap from one-to-one conformance to true interoperability among diverse web services has yet to be realised. In fact, measurable interoperability has yet to be defined satisfactorily, which is why the authors undertook the present study.

The weak composition model described in [3] outlines a declarative composition model for web services, which is centered around descriptive aspects, on interoperability and on scalability. Basically, a web service composition is defined as a set of atomic or compound web services which interact according to certain logical rules. These logical rules specify the patterns of composition (serial, parallel, etc.) which describe the execution order of the services involved in the composition, and the connection flows established for the data flow between services.

[13] See www.opengis.org and follow the link to *Implementation Specifications*.

In comparison with other composition models which are process oriented [4, 5], the approximation presented is essentially oriented to abstract interfaces and uses a backtracking-mode algorithm for the invocation. The resulting composition graph [3], as logical relations between services, is defined implicitly by the very structure of the composition. This graph may be termed a tree structure, whose leaf nodes are linked to atomic services, and whose internal nodes represent virtual intermediate compositions, where the root node represents and encapsulates the entire composition. During invocation this data structure is evaluated by the Interpretation Handler (see section 2) for the orchestrated invocation of the composition's services via the Invocation Handler (see also section 2). Only at execution time are the necessary connections identified and established in order to invoke all relevant services, hence one characteristic that defines the term *weak* composition. Internal nodes with compound descriptions provide both composition patterns and connection flow descriptions.

Weak composition differs from the architecture of traditional workflow management systems because the latter are essentially process oriented in the construction and execution of the composition graph. In existing approximations the developer normally must work with the entire graph, and so complexity increases noticeably as the composition grows. However, the approach described here holds the advantage that in the majority of process models, as complex as they may be, the logical composition graph may be decomposed into basic patterns, serial or parallel, of just two services at a time. The final composition encapsulates the complexity of the model in a manner similar to that of class hierarchy in object oriented languages; the main difference being that here we refer to abstract interfaces and not instantiations of classes.

The reader may be wondering why we have not exploited one of the several existing languages for the composition of web services, such as BPEL4WS, WSCI, BPML, etc. [6]. Within our perceived use case, of chaining 2 or 3 well-known geographic web services—say a Web Map Server to a Web Feature Server—to known e-commerce services such as authentication, is not necessary to overload the system with notions of conditional iteration of web services or feedback among them. Therefore, we consider web services composition without a procedural language, only based on appropriate pattern definitions and the inherent structure of the service description. In addition, there currently exists no international consensus on standard languages for composition; most are merely commercially-driven proposals [7].

Similarly, within traditional flow context, we recognise efforts toward defining standards for workflow interoperability (such as Wf-XML or SWAP) between workflow management systems [8], but this also has not shown a great deal of consensus.

For this reason, we have proposed a new approach different from current languages, and in the context of the goals of the *ACE-GIS* project.

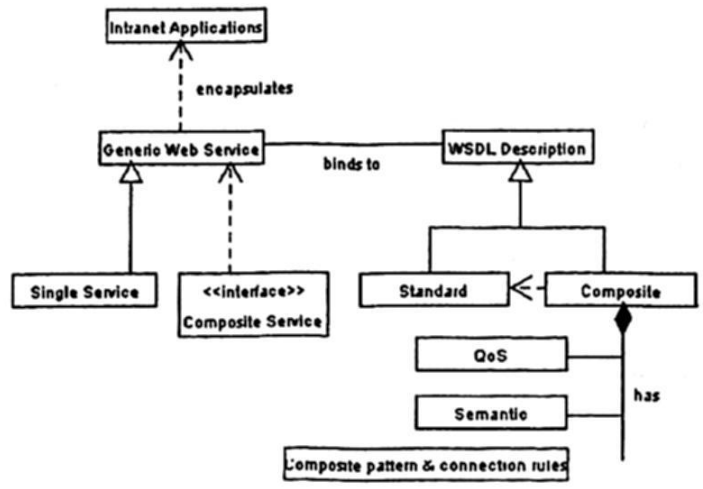


Figure 1. Relations in weak composition (after [9])

A fundamental requisite of organisations (businesses) in the near future [10], will be rapid and dynamic integration of applications and processes to adapt to the web environment. As a first step they often "wrap" applications accessible within the organisation as web services, to facilitate their access by all participants involved in a certain collaborative effort. Current web services technology permits the establishment of loosely coupled connections, which improves interoperability (syntactic at least) among heterogeneous services. At the moment of creating compositions of different available services, new relations appear among the basic elements which form the composition. These relations are shown in the class diagrams in figure 1.

From our viewpoint a general web service can be considered an atomic service or a composition of such general services. In either case the resulting composition should appear to the user as a single entity with which to interact. This affords several benefits including encapsulation of the underlying service complexity and straightforward reuse of a composition in future compositions. The standard manner of describing external behaviour of a web service is via Web Service Description Language, WSDL [11]. Each web service carries with it a WSDL description of the functionalities it offers. In treating a composition as an atomic web service (a so-called opaque service chain) this composition also carries a WSDL describing the entire functionality of the composition. However, in these composition descriptions only the abstract description is present, without details on concrete implementation: things such as access protocols or location points. Service compositions at the abstract level fit comfortably within the weak composition concept, unlike concepts of tightly coupled components within distributed computing models such as CORBA, RMI, etc.

In addition to the abstract WSDL description a composition incorporates the specification of composition patterns and connection flow established for the services

within a composition. These are encoded declaratively in an XML WSDL extension (see Annex for a complete example). Additional aspects such as the quality of service (QoS) desired in a composition, semantic attributes or security considerations, may also be embedded declaratively in the description. This together, abstract WSDL plus XML extension, forms an abstract interoperability interface of the composite service, a basic unit, which is interpreted by the Interpreter and Invocation Handlers (see figure 2) in the model presented. This composite WSDL description (which for the time being we will call WSDL++) is fundamental for the weak composition process, as the approximation follows an abstract component-based model, which facilitates interoperability and connectivity between services.

The remainder of the paper describes in some detail the process of composition and invocation, fundamental building blocks in the conceptual architecture which exploits the weak composition concept. Then we provide a simple example of a prototype implementation of composition and invocation. Finally, we provide conclusions and ideas for future research.

2. CONCEPTUAL ARCHITECTURE

To be able to define the function views that are established in the proposed architecture for composition and invocation of web services, we describe first a global vision of this conceptual architecture detailed in [3].

Basically, the architecture supporting the weak composition concept is composed of 5 layers providing a sort of middleware between the users and the external components such as web services, registries and catalogs:

(i) User layer, formed by applications and interfaces directly accessible by the end user;
(ii) Composition layer, responsible for generating the declarative description of the composition;
(iii) Additional components layer, which contains other components of the composition, such as security, quality of service, semantics or system monitoring;
(iv) Invocation layer, which interprets the composition and executes it;
(v) Local storage layer.

From this conceptual architecture of middleware layers, we establish essentially two functional views of the system, to aid in composition and invocation of services:

(i) Composition view;
(ii) Invocation view.

The composition of services involves primarily components of the user and composition layers. At the user layer are found the components Searcher, Browser and Registry, while the components Composition Handler and Description Handler together form the composition layer of the conceptual architecture. Similar to the composition view, the invocation view contains components pertaining to the user layer, which are assigned to locate, select and navigate among the desired web services. Logically, this view also incorporates the invocation layer, which is necessary for the execution of web services compositions; this layer is comprised of the Interpreter Handler and Invocation Handler components.

The remainder of this section describes in detail the process of composition and invocation of web services, under the guidance of the weak composition conceptual architecture.

2.1 COMPOSITION VIEW

In figure 2 we illustrate a UML sequence diagram outlining the mode in which the search and composition of web services is carried out.

The first 4 steps define the search for available web services, atomic or already compound, supposing the presence of a service catalog accessible locally or remotely (ultimately via web). Once the services are selected, the composition process is initiated (steps 5-14). This composition is supervised by the user with assistance from the Coordinator component. During the composition process, first we establish a composition pattern and connection flow for the messages involved in linking a *pair*[14] of web services (step 5). As mentioned earlier, component-oriented composition consists of decomposing complex compositions in basic patterns of pairs (avoiding the need for a flow language). In this manner the desired composition is constructed in multiple iterations of composing two at a time: for example, simple to simple forming complex, then complex plus another simple, etc., hence the qualifier incremental in the title of the paper. Current composition languages allow construction of several services simultaneously [6], however in our geographic web service context it is not realistic that a user would combine, for example, 10 services at the same time. Therefore, in principle, we prefer a simple and consistent manner to create compositions instead of increasing the cardinality of services composition each time. (This paired process is akin to software debugging, where it makes sense to correct and test one bug at a time. Here we are assured that each paired composition works before we proceed to link another to the composition.) In the prototype application presented the user manually establishes the connections between the two parts (services). One of the immediate goals is to eliminate this

[14] There is no problem in defining patt erns of higher order or even some kind of model language.

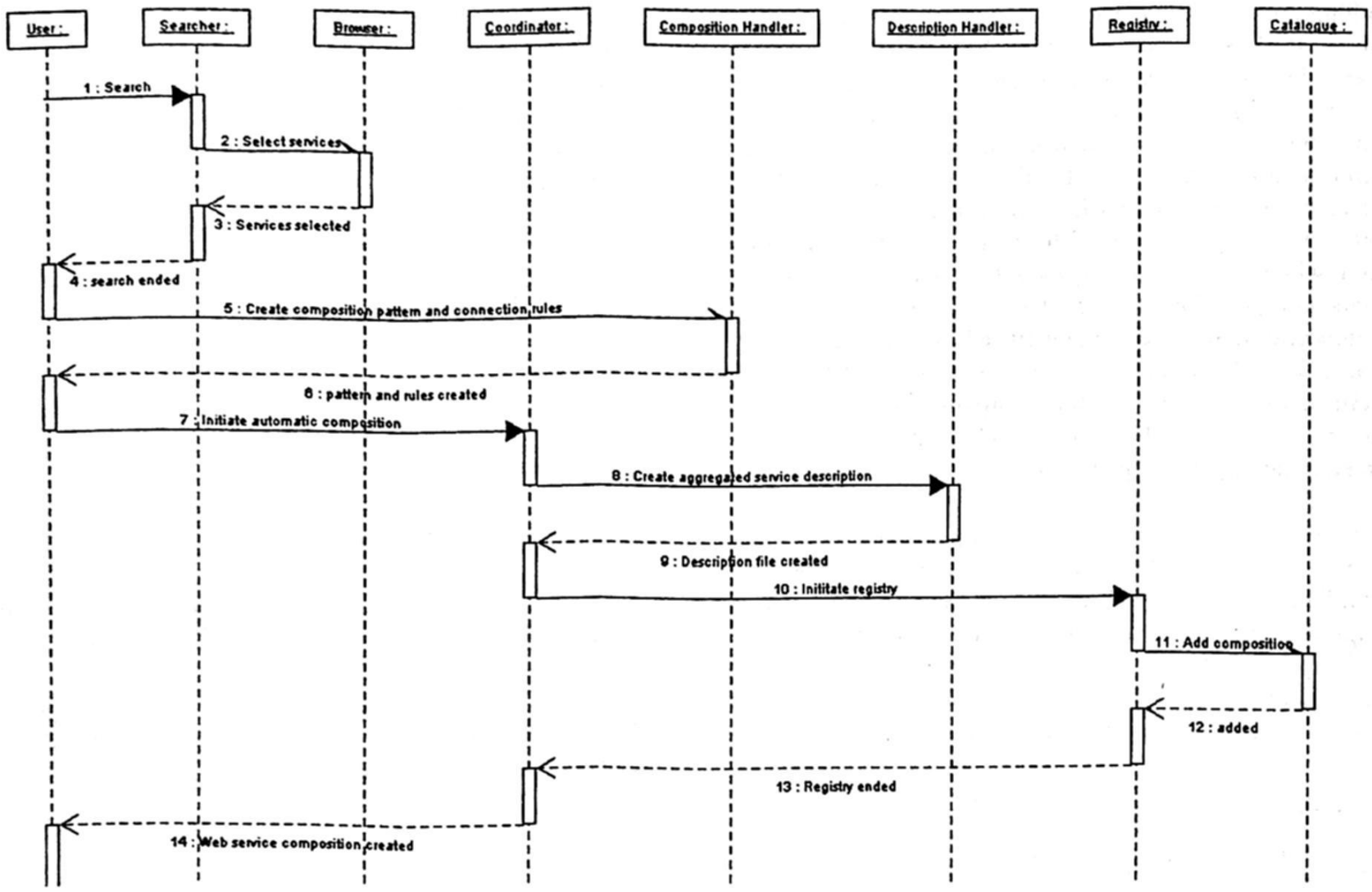


Figure. 2. Sequence diagram for search and composition of services

needs for human intervention to reach increasingly pure levels of automated composition, perhaps initially monitored by the user. This can only be accomplished through the addition of semantic translators, ontology parsers and addition of semantic tags [12], in line with the grand proposal of the semantic web [13]. These extensions are already contemplated within the presented conceptual architecture. For example, a semantic component describing web services and their data types will be in charge of enriching these descriptions from domain ontology (such as for example, transport experts' vocabulary) in order to allow an inference engine to be able to perform "matchmaking", to dynamically detect valid pairs of web services.

The composition process starting at step 7 is automated and controlled by the Coordinator component. During step 8 the Description Handler component encodes the composition patterns and flows in a declarative format based on XML [3]. Later the components from the Invocation layer interpret this encoding at the moment of invocation of the compound service. This format describes two fundamental aspects of the composition: how to connect and in which order to combine the two services, and what should be the external behaviour of the completed composition. The first aspect defines the type of composition, normally serial, and the connection flow for representing the data flow between the two services. The second aspect describes the abstract interface of the two services viewed as a single, independent web service. On this occasion the abstract part defined in the WSDL specification is used. Finally the resulting WSDL++ description of the composition is added to the service catalog, thus exposing the new composition for future use (steps 10-13).

The WSDL++ description of the resulting composition is compatible with the WSDL standard. Any WSDL viewer should be capable of interpreting correctly WSDL++ descriptions produced by our prototype, ignoring the new tags added by our model.

2.2 INVOCATION VIEW

Figure 3 includes a UML sequence diagram which illustrates the mode of search and invocation of both atomic and compound web services.

In general terms the invocation of a composition requires an analysis of the associated description to encode the composition graph as a tree structure. Then, once this structure is created it is traversed in backtracking-mode, invoking the component web services and chaining the results obtained according to the connection flow defined in the WSDL++ description.

As in the composition view, first off the user selects the service or composition he or she wishes to invoke. The invocation process is totally automated and controlled by the Coordinator component. The user need only facilitate

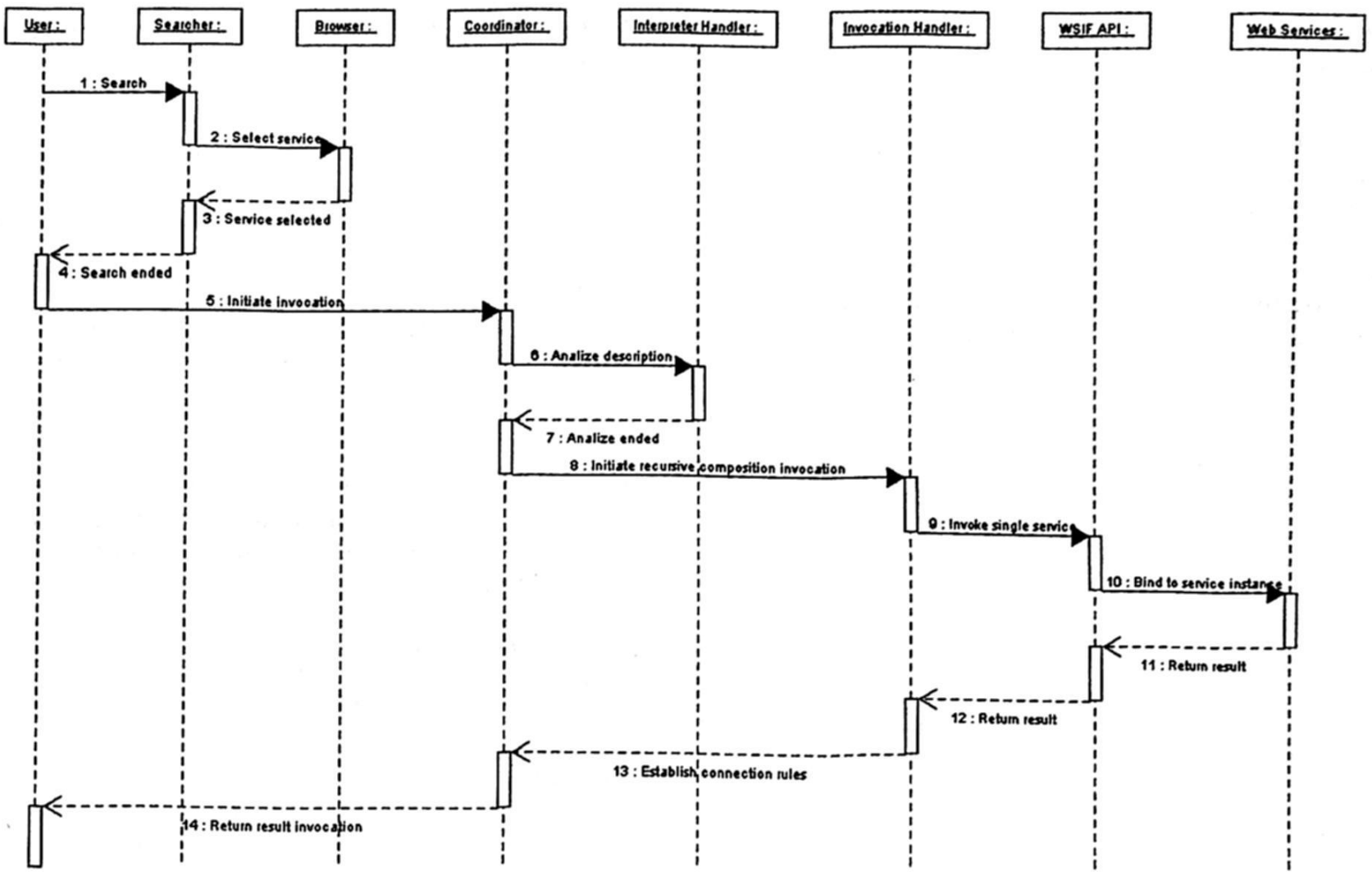


Figure 3. Sequence diagram for search and invocation of compositions

the initial parameters of the composition and await the composition's response (steps 5 and 14). The Interpreter Handler (step 6) realises two basic functions. The first, recursively interpret the WSDL++ of the composition. As all compositions are comprised of services, the Interpreter Handler analyzes the composition description in search of the services which compose it. If it encounters atomic services then the search halts and no further analysis is needed. If, however, it encounters compound services, the Interpreter Handler continues analyzing the new composition until it reaches a pair of atomic services.

This analysis algorithm extracts the composition graph from the very structure as defined by the composition description. The second function of the Interpreter Handler is the creation of the tree structure as an implementation of the composition graph. In this interpretation process we can appreciate how the composition graph is implicitly defined by the composition structure. As a consequence, at no moment does the user need to construct the complete graph at design time in order to create the desired composition.

Once the tree is created, the Invocation Handler component takes control to initiate the invocation process (steps 8-13). Invocation is a backtracking process where the leaf nodes are directly invoked as atomic web services (step 10) via the Web Services Invocation Framework (WSIF) [14], and open-source API provided by the Apache Software Foundation (www.apache.org) which permits the invocation of web services in manner which is independent of protocol or access method (SOAP, HTTP GET/POST, etc.). Internal nodes on the tree are not directly executable because they describe virtual compositions. Their mission is to interpret the composition patterns and connection flows of the children nodes, that is, decide whether to execute them in series, parallel etc. and how to establish the data flow between these children nodes.

The current implementation of our model executes the service composition via the invocation of each instance and connects the information between services as a function of the data flow established by the user. This vision of static composition has availability problems if one of the services constituting the flow is not operational or available at the moment of execution. Future investigations are aimed at dynamic services composition at execution time, whereby the composition itself is able to detect functional anomalies and can substitute defective services with alternates in a transparent manner.

Aside from the mentioned limitations, backtracking-mode invocation through the weak composition model achieves a dynamic binding with concrete web service instances at execution time. The access mechanism to a service is not known until the moment of invocation. In this way, the composition using the abstract interfaces provides maximum flexibility compared to binding at design time of the composition.

3. AN EXAMPLE: ARITHMETIC FUNCTIONS COMPOSITION

In this section we present a web based prototype application which has been developed for the rapid integration, composition and invocation of web services, in an incremental fashion. Utilization of this prototype permits us to test and demonstrate a first approximation of the component based composition described here, its pros and cons, as well as the backtracking process in the composition invocation.

Following an interesting example of human arithmetic composition during World War II, described by Feynman[15] [15], we compose functions based on a chain of basic arithmetic operations, each exposed as a web service.

This simple example is illustrative in order to demonstrate the capabilities of the prototype application. Future work, already under way, is to apply the prototype to geographic web services, beginning with well-known services such as those defined by Open GIS Consortium, such as a composition of a Web Map Service and a Web Feature Service. While these tests will be far more realistic and useful to an end user, they pose a degree of complexity which does not help us to explain the basic concepts in a didactic manner here.

Our simple example is based on the four basic arithmetic operations (addition subtraction, multiplication, and division) each of which is exposed as a separate web service. Each web service implements an operation through a single public method of the type:

Add (op1, op2) : AddReturn
Subtract (op1, op2) : SubtractReturn
Multiply (op1, op2) : MultiplyReturn
Divide (op1, op2) : DivideReturn

Imagine that we wish to solve compositions of arithmetic functions. For example if f y g are combinations of basic arithmetic functions, then $f \otimes g$ would be a valid composition of functions. The function f might be defined as the combination of a sum and a division, whereas the function g might be another combination of the operations subtraction and multiplication:

$$f(x,y,z) = \frac{x+y}{z}$$

$$g(u,v,w) = (u-v)*w$$

In this manner the composition $f \otimes g$ would take the following form:

$$f \otimes g = f \otimes g(x,y,z,v,w) = g(f(x,y,z),v,w) = ((\frac{x+y}{z}) - v) * w$$

If we analyze the resulting composition, it would be formed by a combination of simple arithmetic services. Evidently, it would be totally inefficient to attempt to create a separate and permanent web service for each possible arithmetic composition. In this case no service exists to resolve $f \otimes g$, however exploiting the combination of existing simple services it becomes possible to meet the special needs of the desired combination.

The remainder of this section describes in some detail how we realize the composition and invocation of arithmetic functions using the prototype which implements the weak composition model.

3.1 COMPOSITION OF ARITHMETIC FUNCTIONS

The construction of compositions in the weak composition model follows an incremental, top-down design, that is to say it is initiated with the union of atomic services to form intermediate compositions, until arriving at the compound compositions desired by the user. In order to create the composition $f \otimes g$, we previously would need to have created the intermediate compositions f y g. Consequentially, there are three new compositions awaiting in a registry, to be reutilized at another moment or by other users: f, g and $f \otimes g$.

Figure 4 shows the component Composition Handler for the creation of f at the moment of specifying the composition pattern and connection flow. This component is divided in two zones. The upper part shows information relevant for the user of two services to be combined, as for example the *Add* and *Divide* operations. In the lower part, the more interesting part, the composition pattern and connection flow for the two services are specified. Specifically, the output parameter *AddReturn* of the *Add* operation with the input parameter *op1* of the *Divide* operation. The other input parameter of *Divide*, *op2*, is defined as a parameter external to the composition that is, introduced by the user at the moment of invocation.

Additionally, in this prototype it is possible to define input parameters as constants, whose value remains embedded in the WSDL description of the composition. Once the parameters defining the composition are configured, the Description Handler component creates the associated WSDL++ description and adds it to the registry of available services (see Annex for the complete WSDL++ description of composition *f*).

[15] Nobel prize-winning physicist Fe ynman describes how, during the Manhattan Project in Los Alamos, he formed 'chains' of dozens of female data processors, each armed with a mechanical calculating machine and assigned to receive a punch card from the previous person, perform a single mathematical operation on it --addition, subtraction, cube root, etc.--and then pass it to the next person.

http://localhost/services/CompHandler - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Web Services Composition Table

Information of Web Services

Left Web Service		Right Web Service	
Operation: Add		**Operation: Divide**	
Input message	**Output message**	**Input message**	**Output message**
AddRequest	AddResponse	DivideRequest	DivideResponse
(op1, *type xsd:string*) (op2, *type xsd:string*)	(AddReturn, *type xsd:string*)	(op1, *type xsd:string*) (op2, *type xsd:string*)	(DivideReturn, *type xsd string*)

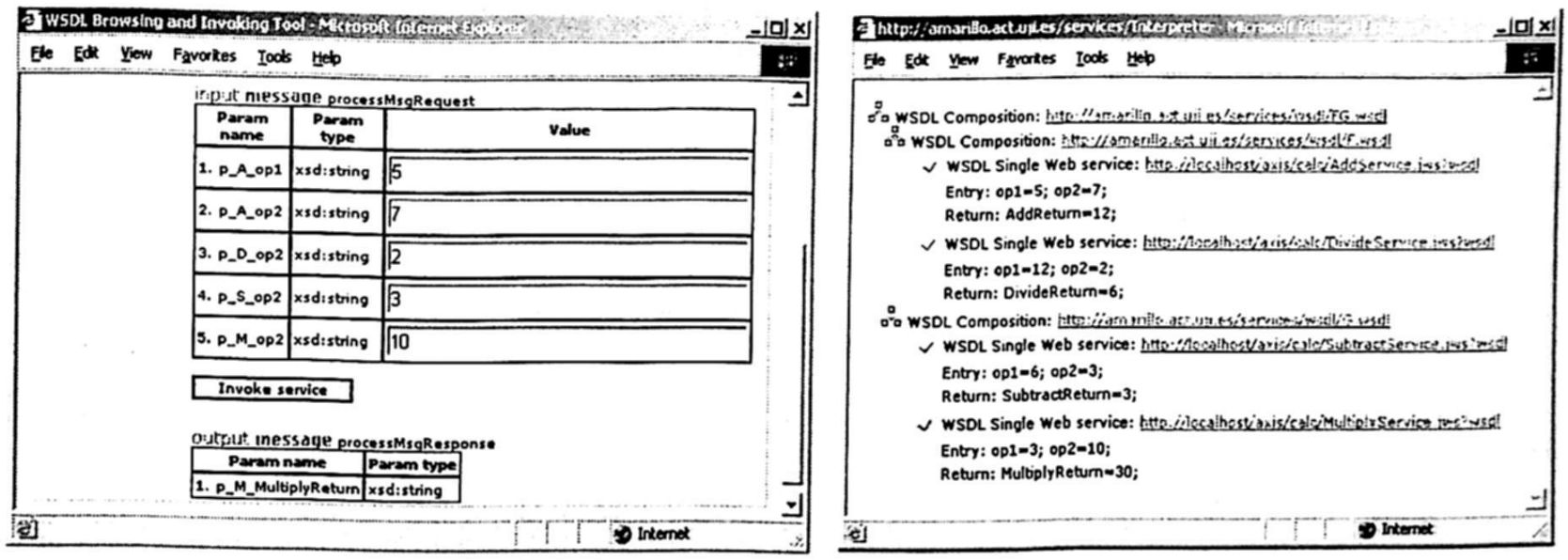
Composition Pattern - Connection Flow

Aggreg. WSDL name	Composition pattern	Connection flow	Actions
F	Sequence Parallel	Output Parts ▾ External ▾ Input Parts ▾	Add Connection Create WSDL++

AddReturn *is connected to* **op1**
op2 *is external parameter*

Done Local intranet

Figure 4. Specification of a composition pattern and th e connection flow for the creation of composition f.



(a) (b)

Figure 5. (a) - User sets initial parameters for $f \otimes g$ composition invocation; (b) - Result for $f \otimes g$ composition invocation.

In the same way, the user creates a composition g and then the final composition $f \otimes g$, from the intermediate compositions f and g. This iterative method of gradual composition encapsulates the complexity of the composition at design time. It is far easier for the user to compose $f \otimes g$ from f and g than from the basic arithmetic operations themselves.

Also, the incremental composition model alongside the possibility to independently invoke each intermediate composition created, facilitates debugging a priori. Utilizing this prototype it is possible to create each intermediate composition, execute it, and validate it correct operation. This characteristic helps save effort in later debugging steps in the final composition.

3.2 INVOCATION OF ARITHMETIC FUNCTIONS

The invocation process is centred on tree structure for the composition graph. The general invocation process of any composition is comprised of the following steps. The user introduces the initial composition parameters. Then the components of the Invocation layer are charged with creating the tree based on the description of the composition, with invoking the composition while establishing the data flows between connected services. Finally the component Coordinator returns the resulting data to the user.

Returning to the example of invocation of the composition $f \otimes g$, the user introduces the initial parameters of the composition (see figure 5a). Next, the Interpreter Handler is responsible for creating the tree structure. During the process of analysis the Interpreter Handler creates a node for each composition or atomic web service encountered, establishing double links between a father node and its children nodes. The initial invoked composition is identified by the root node. Each of the atomic services or intermediate compositions forming a given composition is converted into a child node of the node identifying the main composition. Specifically, the composition $f \otimes g$ is converted in root node and the compositions f y g as children nodes on the left and right respectively. Continuing the analysis recursively, the node associated with the composition f contains as children the atomic web services *Add* and *Divide*, while the node associated with the composition g contains the services *Subtract* and *Multiply*.

Once the tree is constructed, we need only to invoke it to obtain the composition's result. It is not possible in all cases to establish a priori the concrete value of the data flow between the services comprising the composition. Normally dependencies at execution time will impede the specification of concrete values for all parameters of web services. It is necessary to utilize a backtracking mode algorithm to traverse the tree, in order to realize simultaneously the downward search and adjustment of inter-service data flow while returning. For example in invoking composition f, the services *Add* and *Divide* are defined in sequence. In this case it is not possible to begin execution of the service *Divide* without first obtaining the results of *Add*. In figure 5b we find the result of invoking the composition $f \otimes g$. In it we see the hierarchical structure of the invocation of the tree which represents the logical composition graph.

4. CONCLUSION AND FUTURE WORK

In this paper has been presented a model for weak composition of web services within the context of the *ACE-GIS* project [2], on-going research on geographic web service interoperability.

The model proposes a definition of web service compositions which are weakly or loosely coupled based only on abstract syntactic descriptions in WSDL. This weak composition is specified declaratively in XML, WSDL++, as an aggregation of external descriptions of the services as well as the composition pattern and connection flow.

From the inherent structure of this weak composition the Interpreter Handler generates a tree structure which the Invocation Handler then traverses in order to invoke the compound service.

With this model we attend to several aspects of the service composition problem:

(i) Compositions are specified based on abstract descriptions of the services, linked to WSDL interfaces. This permits great flexibility in the integration of services which are *a priori* incompatible.

(ii) Dynamic coupling of service instances is accomplished at execution time. The mechanism for accessing services is not known until the moment of invocation. This results in flexibility compared to design-time coupling.

(iii) The oriented abstract interfaces composition and the backtracking-mode invocation of services via the logical composition graph are features of the proposed model. It permits that both compositions and simple web services may form a part of future compositions, which take on the appearance of atomic services.

(iv) The construction of compositions follows an incremental design which hides composition complexity from the user (encapsulation), facilitates independent invocation of intermediate combinations, and facilitates the debugging and error detection in the final composition.

(v) The combination of patterns and the inherent structure of the composition eliminate the need

for a flow language. This main characteristic differentiates the weak composition concept from current languages for the composition of web services.

(vi) This model defines a framework in which semantic aspects; quality of service, service cost, etc. could be included.

Thus far we have developed a prototype capable of guiding the user in the creation of compositions formed by an arbitrary number of existing web services. This prototype demonstrates the characteristics defining weak composition that is, permitting the composition of web services based on principles of interoperability and scalability.

As far as future work, we are labouring within the *ACE-GIS* project to compose a simple emergency management system combining a Web Feature Service (gas release plume model) and a weather service (reporting wind conditions at a given location); this is the logical next step toward supporting diverse geographic services. Additionally we are working with *ACE-GIS* partners (University of Münster) to incorporate semantic aspects (extensions of DAML-S) of the emergency management situation, with the goal of improving semantic interoperability within the composition. Finally, to help assure that the resulting service composition is somewhat fault tolerant with regard to service availability and quality of service, we are also investigating dynamic composition at execution time [16], so that the composition will be able to detect service faults and replace faulty services with substitutes meeting user requirements.

5. ACKNOWLEDGEMENTS

The work described here has been partly supported by the projects TIC-2000-1568-C03 (Spain Ministry of Science and Technology), IST-2001-37724 (European Union) and CTIDIB/2002/336 (Valencia government: Generalitat Valenciana).

REFERENCES

[1] M.P. Singh, Physics of Service Composition, *IEEE Internet Computing, 5*(3), 2001, 6-7.

[2] M. Gould, J. Poveda, & J. Huerta, ACE-GIS: Integración y Composición de Servicios Web Geográficos y de E-comercio. In: Toro, M. (Ed.): *Proc. II Jornadas de Sistemas de Información Geográficos*, El Escorial, Madrid, November 2002, 7-17. Project described in English at http://www.acegis.org

[3] C. Granell, J. Poveda, & M. Gould, Composición Débil de Servicios Web, *Submitted to professional journal.* Also available on-line at: http://www3.uji.es/~canut/weakcomposition.pdf

[4] D. Fensel & C. Bussler, The Web Service Modeling Framework WSMF, *Electronic Commerce Research and Applications Journal, 1*(2), 2002, 113-137.

[5] B. Benatallah, Q.Z. Sheng, & M. Dumas, The Self-Serv Environment for Web Services Composition, *IEEE Internet Computing, 7*(1), 2003, 40-48.

[6] S. Aissi, P. Malu, & K. Srinivasan, E-Business Process Modeling: The Next Big Step, *IEEE Computer, 35*(5), 2002, 55-62.

[7] W.M.P. Van Der Aalst, Don't Go with the Flow: Web Services Compositions Standards Exposed, *IEEE Intelligent Systems, 18*(1), 2003, 72-76.

[8] J.G. Hayes, E. Peyrovian, S. Sarin, M.-T. Schmidt, K.D. Swenson, & R. Weber, Workflow Interoperability Standards for the Internet, *IEEE Internet Computing, 4*(3), 2000, 37-45.

[9] B. Benatallah, M. Dumas, M.C. Fauvet, & F.A. Rabhi, Towards Patterns of Web Services Composition. In S. Gorlatch and F. Rabhi (Eds): *Patterns and skeletons for parallel and distributed computing* (UK: Springer Verlag, 2002).

[10] IDC analysts, Web Services Adoption Timeline and Related Business Opportunities, 2002. http://www.idc.com/en_US/st/extras/webServices.pdf

[11] R. Chinnici, M. Gudgin, J.-J. Moreau, & S. Weerawarana, Web Services Description Language (WSDL) Version 1.2: *Core Language. W3C Working Draft 11 June 2003.* http://www.w3.org/TR/wsdl12/.

[12] E. Sirin, J. Hendler, & B. Parsia, Semi-automatic Composition of Web Services using Semantic Descriptions, *Proc. of Web Services: Modeling, Architecture and Infrastructure Workshop at ICEIS 2003*, Angers, France, 2003.

[13] T. Berners-Lee, J. Hendler, & O. Lassila, The Semantic Web, *Scientific American, 284*(5), 2001, 34-43.

[14] Web Services Invocation Framework, Apache Software Foundation, 2003. http://ws.apache.org/wsif/.

[15] R.P. Feynman, *The pleasure of finding things out: The best short works of Richard P. Feynman* (Cambridge, Massachusetts: Perseus Publishing, 1999).

[16] S. Ghandeharizadeh, G.A. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J.L. Ambite, M. Cai, C.-C. Chen, P. Pol, R. Schmidt, S. Song, S. Thakkar, & R. Zhou, Proteus: A System for Dynamically Composing and Intelligently Executing Web Services. In the *First International Conference on Web Services (ICWS)*, Las Vegas, Nevada, June 23-26, 2003.