To appear in EC-Web 2005. Copenhagen (Denmark), August 2005

Improving Reuse of Web Service Compositions

Carlos Granell[1], Michael Gould[1], Roy Grønmo[2], David Skogan[2]

[1] Department of Information Systems (LSI)
Universitat Jaume I, E-12071 Castellón, Spain
`{carlos.granell,gould}@uji.es`
`http://www.geoinfo.uji.es`

[2] SINTEF Information and Communication Technology
N-0314, Oslo, Norway
`{roy.gronmo,david.skogan}@sintef.no`
`http://www.sintef.no`

Abstract. We describe a methodology for assembling composite services based on three basic processes which are independent of the concrete implementation: Service Abstraction Process, Service Composition Process, and Translation Process. These processes share the concept of integrated component composed of two key aspects: a specific set of the Aalst's workflow patterns together with a component-style composition of complex services. We propose a novel approach that implements the steps of such methodology, providing an efficient manner for developing service compositions and enhancing the expressiveness of target composition languages like BPEL4WS. Here we focus on the description of the Service Abstraction Process, a critical step in order to enhance the service composition by facilitating the reuse of existing services.

1 Introduction

In the service oriented architecture field, several different approaches and platforms are being developed to address the common goal of web service composition [8]. Each provides unique perspectives to facilitate service integration, from static and dynamic approaches to manual or automated ones, and each approach defines its own manner for composing services, provides its specifications and languages, and introduces new specifications to the web service protocol stack[1]. In fact each of the possible service facets (addressing, discovery, context, etc.) normally carries a specification or even two or more potential specifications with no single consensus standard defined to date, leading to overlapping features [13]. This may lead to intercompatibility issues among different approaches, complicating even further the service composition problem. In this work, rather than provide a new specification from the ground up, we prefer to build our approach, where possible, upon existing well-known web services specifications or *de facto* standards (like for example HTTP, WSDL, and BPEL).

[1] http://roadmap.cbdiforum.com/reports/protocols/

Most similar approaches attempt to address service composition by composing *single* web services from scratch, ignoring reuse of existing compositions or parts of compositions. In this sense, from a developer's perspective it is interesting to explore the reuse of existing services. It is expected that this higher level of service reusability will lead to more efficient, more structured composition process that will accelerate rapid application development.

In this paper we concentrate on how the service reuse can be exploited in the service composition process for developing more rapid and efficient web applications. First we describe a service composition methodology for developing applications composed of three steps, namely Service Abstraction Process (SAP), Service Composition Process (SCP), and Translation Process (TP). The SAP is charged with distinguishing abstract services from concrete services, ready for composition. The SCP provides the needed means for creating service composition reusing existing ones. At run-time, the TP translates the service composition description into a target composition language like BPEL4WS to be executed by a workflow engine. Secondly, we propose a service composition model using specific workflow patterns and a component-style composition of complex services. The set of workflow patterns is based on [1], adapted to the service composition context. The *integrated component* notion incorporates component aspects into web service concept. Integrated components together with workflow patterns become the fundamental blocks for application development by reusing and combining simpler components into more complex ones. In this paper we focus on the description of the SAP process within our model, the proposed workflow patterns set and its relationship to the integrated components.

The remainder of the paper is organized as follows. The proposed methodology is described in section 2. Section 3 presents the SAP process in detail, describes the integrated component concept, workflow patterns used, and how to design these integrated components on our model. Finally, section 4 concludes the paper.

2 A Service Composition Methodology

Our objective is to introduce a conceptual methodology supporting service reuse and composition. It consists of applying the following basic processes: the SAP for creating abstracts services or integrated components, the SCP for constructing complex applications (composite services) reusing abstract services, and the TP for converting complex services into a target composition language for deployment and invocation. The rest of this section describes each process presenting its purpose, the modules implicated, and their relationships.

2.1 Service Abstraction Process

The core of service-oriented architectures (SOA) is essentially collections of services (operations) described by an interface and provided to a user or client. Composing web services by means of *concrete* service interfaces leads to tightly-coupled compositions in which each service involved in the chain is tied to a web service instance. If current services are replaced by new or updated ones, changes in the underlying workflow may become necessary from, for example, slight modifications of bindings to wholesale redesign of part of the workflow description. Because of that, services should be interpreted as abstract notions facilitating their independent composition. Figure 1 depicts a UML class diagram illustrating the elements and their relationships contained in the SAP.

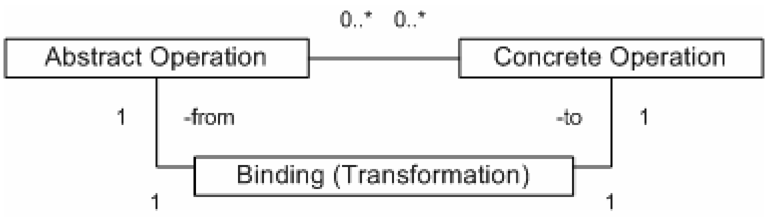

Fig. 1. Elements within the Service Abstraction Process

The SAP is a process (abstraction process) which allows us to move service operations from terms of particular functions to shared and agreed domain concepts. In particular, ontologies and semantic properties are a basic ingredient in the abstraction process because these introduce meaning to services (for example semantic properties) and thus enable the reuse of these services and their independent composition. A minimum required attribute for describing the service functionality is its signature, composed of its operation name and its input and output parameters. The developer (or software module) should be able to represent different operation signatures as a (new or existing) single abstract signature, specified as the minimal signature required for a given functionality [6]. In this way, the resulting abstract operation associated with each domain forms the basis for the future composition process, which is carried out in terms of common abstract signatures instead of concrete signatures. From Figure 1, the SAP may be composed of the following modules:

- *Abstraction module.* This module is charged with abstracting operations: to map operation and parameter names to comprehensible names defined by means of common vocabularies or taxonomies. The resulting abstract operation is made available in service registries. In fact, every abstract operation is actually a set of candidates operations with a given functionality, of which the execution results of just one will be considered at run-time by a workflow engine. We will ignore aspects concerned with the quality of service [7] although we realize all these properties – security, availability, efficiency, response time, etc. – play an important role for selecting the most appropriate operation for execution among the candidates.
- *Binding module.* The abstraction process itself includes a gap between the abstract specification of an operation and how to access to an operation on a technical level. The binding description specifies how to map from abstract operations to concrete operations. The established bindings are meant for service invocation and should be kept separate from the description of the abstract and concrete operations, the three logical views of an operation - abstract, concrete and binding - remaining physically separate.

2.2 Service Composition Process

A composite service incorporates the business logic and functionality of several simpler services contained within [2]. The problem addressed is how to build and integrate such a composite service by reusing other simpler services.

A natural way of conceptualizing service composition is by means of composing single and composite services recursively [2, 8]. In this way, a composite service is defined as an aggregation of other single and composite services. The SCP suggests an extension of this idea by adding two new aspects. First, component technology [12] allows modular composition of software systems from reusable and independent code described by explicit interfaces. In essence, this approach is identical to others carried out within different contexts like for example component-based software development [11], in which the application logic is constructed by composing smaller software components. Therefore, the *composition process* constructs complex applications as integrated components by incrementally aggregating and reusing existing ones. Second, a specific set of workflow patterns defines the orchestration among the integrated components. This provides an added level of simplicity, independence, and reusability in the composition process.

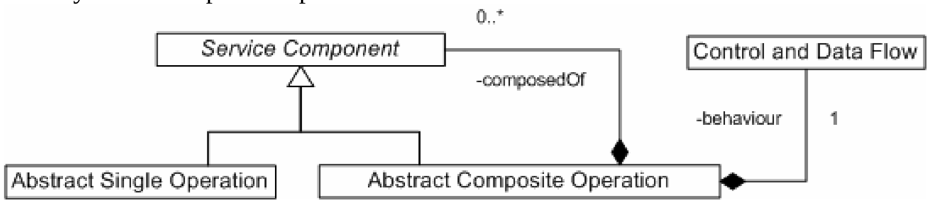

Fig. 2. Elements within the Service Composition Process

As depicted in Figure 2, the relationships among the tasks in a composition process starts from the level of abstraction defined by abstract single operations, which are previously created by the SAP. An abstract single operation is considered a basic element. An abstract composite operation is in turn an aggregation of other abstract (single or composite) operations, which are referred to as *service component*. The service component may comprise single or composite operations and specify the orchestration among the contained operations (service components), that is, the control and data flow description. The control flow establishes the partial order in which the service components should be invoked. The data flow captures the data dependences among such service components. Based on these relationships, the SCP may be composed of the following modules:

- *Composer module*. It handles the composition process by combining service components from single or composite ones. The resulting service component is also considered as an integrated component. Orchestration features among contained service components are also specified by this module. Therefore, the composition process consists of discovering existing abstract operations, designing the resulting composition, and registering it for future use.
- *Discovery module*. This module is concerned with selecting the required service components for composition.

2.3 Translation Process

Once a composition is created following the two processes mentioned, we use the TP to transform the composition into a target web service-oriented workflow language to permit users and external programs to execute it. The TP converts the integrated component(s) description to a workflow language (for example BPEL4WS) while also enhancing the expressiveness of these languages since the target process description is reached by following a component-style composition.

All these combined processes provide a methodological, an efficient way for developing applications by service composition and reuse. Users may create customized applications following the service composition methodology in three steps, using the underlying concept of integrated component with regard to its creation, composition, and translation: they first create abstract services as integrated components. Then, application development is performed by incrementally aggregating existing integrated components. Finally, these complex applications created are translated into executable descriptions for invocation.

3 Abstraction Process for Designing Integrated Components

Now that the service composition methodology has been introduced, we describe the abstraction process according to the SAP. The key elements are integrated components and workflows patterns. In section 3.1 we propose the integrated component concept for expressing services. Section 3.2 presents appropriate workflow patterns for selection of services within an integrated component. Finally, section 3.3 discuses the modeling of services based on such integrated components and workflow patterns.

3.1 The Role of Integrated Components

Expressing services as blocks of components is especially attractive within service composition contexts for different reasons. First, by definition, components are the sole ingredients of the composition process [12], which is a critical aspect to simplify the composition model as a useful and practical process. If we treat both single and composite services as components, recursive composition becomes an implicit and natural means for building complex services (compositions). Next, components must be units of replacement [12] avoiding external data dependencies. This means that a component is not dependent on the composition in which it is involved. It can be reused in others when the functionality that it exposes is required. Furthermore, it is clearly relevant for improving service reuse that components incorporate no implementation details, only metadata describing all service aspects involved in the composition process.

We adopt a component-based approach to model web services (earlier versions of this concept are found in [3, 4]). We define an *integrated component* as a service adopting the component technology's aspects explained above. It comprises either abstract single operations (SAP's outcome) or abstract composite operations (SCP's outcome). We consider either abstract (single or composite) operation is an integrated component. To accomplish that, all service aspects relevant for service composition need to be captured in abstract descriptions. From the set of layers for designing a service [10], we take into consideration the following service aspects for enabling integrated components: (i) *descriptive aspects* are metadata concerned with the context in which the service operation is performed; (ii) *functional aspects* detail the service capabilities in terms of operations, parameters, etc., that is, its functionality; (iii) *structural aspects* show how a service is internally structured as a combination of simpler services; (iv) finally, *binding aspects* establish relationships between abstract and concrete specifications required for service invocation.

An integrated component combines the component and services aspects into the single notion, maintaining the benefits of both, by two functional interfaces: public and private. The public interface expresses publicly the service's descriptive and functional aspects. The private interface represents an internal view of the integrated components encapsulating the structural features like control and data flow as well as the necessary transformations for binding.

3.2 Workflow Patterns

Workflow patterns also play a key role in our model because they describe the structural features of an integrated component. Firstly, this section selects a relevant set of workflow patterns for service composition. Then, it discusses which of those selected workflow pattern sets are applied in the abstraction process and their relationship with the integrated components.

The original 20 workflow patterns [1] provide a framework for comparing different workflow management systems according to their functionality (see Table 1). However, not all workflow patterns are relevant to the service composition context. Our choice is based on simplicity criterion where *sequence* (1), *parallel split* (2), *choice* (4, 6), and *replication* (9) constructs offer enough means to model complex patterns in any composition [9]. Counterpart patterns regarding synchronization (3, 5, 7, 8) are also significant in our model since the split and join combination aligns perfectly to the integrated components' hierarchical structure for building composition as stated in the previous subsection.
The remainders of the workflow patterns (10-20) are not considered for service composition. Some irrelevant workflow patterns (10, 12, 13, 17) can be modeled with basic ones (Table 1, left column). For instance *multi-merge* (10) defines several parallel executions with a specific joining point can be represented by using *and-split* (2) and *and-join* (3) together. Others are not suitable for service composition as *deferred choice* pattern (16) since it assumes an external input in the workflow. As described in section 3.1, we prefer integrated components with lower levels of external data dependencies. The rest of workflow patterns (11, 14, 15, 18, 19, 20) have a great importance on the execution of the workflow, however they have no relevant meaning during the abstraction process described in this section.

Table 1. Relevant and non-relevant workflow patterns (synomymous term in parenthesis)

No	Relevant Workflow Patterns	No	Non-Relevant Workflow Patterns
1	Sequence (*seq*)	10	Multi-merge
2	Parallel split (*and-split*)	11	Implicit Termination
3	Synchronization (*and-join*)	12-13	Patterns involving Multiples Instances (design time)
4	Exclusive choice (*xor-split*)	14-15	Patterns involving Multiples Instances (run time)
5	Simple merge (*xor-join*)	16	Deferred Choice
6	Multi-choice (*or-split*)	17	Interleaved Parallel Routing
7	Synchronizing merge (*or-join*)	18	Milestone
8	Discriminator (*disc-join*)	19	Cancel activity
9	Arbitrary circles (*loop*)	20	Cancel Case

Understanding how integrated components are related to workflow patterns is critically important since the composition process is generally considered to be the application of a composition operator [12]. In our model, workflow patterns play the role of *selection* operators in the SAP and of *composition* operators in the SCP. Here we focus on the selector role during the abstraction process.

Each and every resulting abstract operation, the abstraction process' outcome, is considered an integrated component. An integrated component comprises a set of candidate operations with similar functionality. An appropriate workflow pattern is needed to execute one operation from a potential operation set. The idea is to extend the functionality of the *alternative services* pattern proposed in [5]. Suppose that one service is selected for execution but that such a service becomes unavailable due to server or network troubles, failing the workflow execution. To prevent this, the alternative services pattern allows us to model alternative services in the workflow that perform the same task as the most appropriate service selected by quality, data or simply user preferences.

Figure 3 illustrates the service selection situation. Both the split and join conditions are needed for modeling the service selection into a target composition language. The split condition is concerned with the candidate operation set (inputs) whereas the join condition refers to the selected operation (output). From the relevant workflow patterns in Table 1, the inputs `or-split` and `and-split` are suitable whereas for the output we are interested in `disc-join` pattern because only one of them will be finally considered by the workflow engine. Therefore, the `or-split` with `disc-join` and `and-split` with `disc-join` pairs are appropriate in the abstraction process. The former is suitable when the user selection explicitly includes some conditions (for example, quality or pre-conditions) over the candidate operations taking them into account *a priori*. This selected set is modeled for execution [5] and only the first one to terminate successfully is considered and the others are ignored at run-time. We can say that two filters are applied: the first one concerned with criteria at design-time and the second one concerned with service availability at run-time. For the latter all candidate operations are considered for execution. In this case, only the service availability filter runs. Therefore, these selection workflow pattern pairs are reflected in the resulting workflow description to help assure that the composition is somewhat fault-tolerant due to multiple service availability.

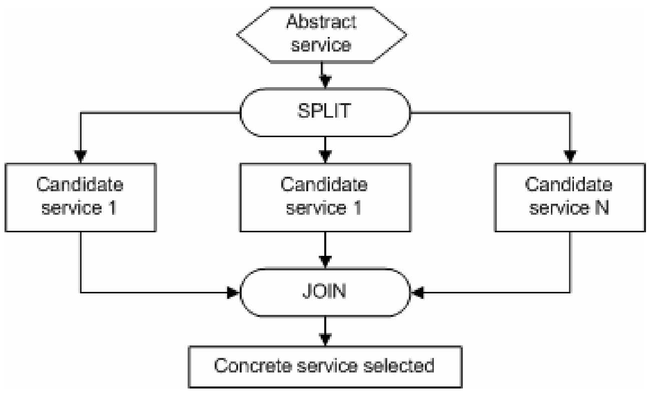

Fig. 3. Workflow pattern for service selection in the SAP

3.3 Modelling Integrated Components

Building on the concept of integrated components in section 3.1 and the analysis of workflow patterns in section 3.2, here we detail the abstraction process for modelling integrated components.

The objective of this abstraction process is to build integrated components which will then be used for the composition process within the SCP. From the user perspective the public interface, the integrated component's functionality and descriptive aspects, is interesting and not how such a component is internally constructed. From the programmer or designer perspective, the private interface is also interesting because it shows how the component is internally structured. Therefore, modelling integrated components consists of specifying all service aspects stated earlier.

Figure 4 summarizes the major steps in the abstraction process in which a bottom-up design is necessary to reach the abstraction operation level of available concrete operations: (i) the definition of the required abstract operation in terms of is domain; (ii) the functionality associated with the abstract operation; (iii) the internal structure definition; (iv) the definition of the appropriate transformation between the abstract and concrete operations; and finally (v) the abstract operation registration. An integrated component is generated by steps (i) and (ii) which represent the public interface and by (iii) and (iv) which correspond to the private interface.

The first step (i) consists of finding an appropriate abstract operation that corresponds conceptually to a given concrete operation. At this moment two possibilities are possible: either an abstract operation already exists or does not. For the former, we specify the descriptive aspects of this new abstract operation which can be, for example, commerce classification, service category, or domain, such as are represented in an OWL-S service profile. For the latter, such a concrete operation is actually being assigned to an existing abstract operation then specifying the descriptive and functionality aspects are not necessary. Indeed, it is likely that several operations within a similar domain will belong to the same integrated component. For example, let us consider a couple of operations concerned with weather information (`Get Wind` and `Get Weather Report`), where the `Get Wind` operation returns

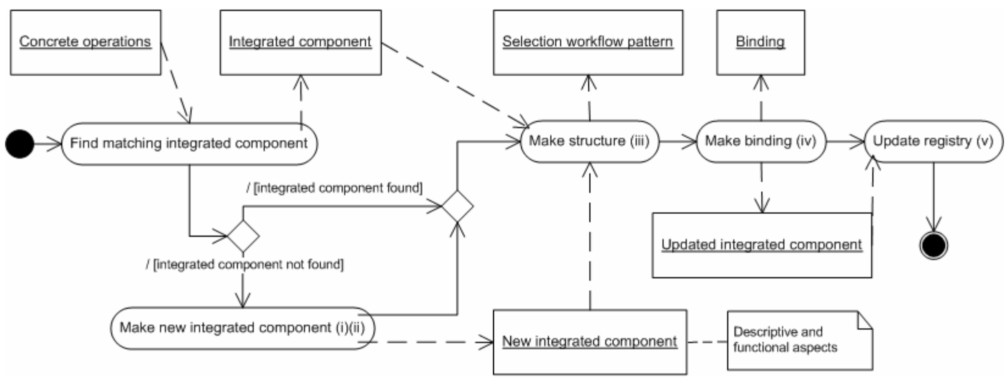
Concrete operations
Integrated component
Selection workflow pattern
Binding
Find matching integrated component
Make structure (iii)
Make binding (iv)
Update registry (v)
/ [integrated component found]
/ [integrated component not found]
Updated integrated component
Make new integrated component (i)(ii)
New integrated component
Descriptive and functional aspects

Fig. 4. UML activity diagram for the abstraction process in the SAP (numbers in parenthesis correspond to service aspects described in section 3.1)

only certain wind-related variables a complete weather report, including in addition to wind information, weather forecast features such as pressure, humidity, or temperature. In this case, the integrated component (`Wind Info`) should be formed by the minimal common information among them, hiding both operations under the same domain (the `Wind` concept). Similarly, in a complex operation like `Get Weather Report`, some parts of the same function may belong to several integrated components with different concepts (`Wind, Pressure, Humidity`, and `Temperature` concepts). Both service availability and reuse are improved by decomposing a given complex operation into its finer functionalities, leading to multiple logical views of the same operation. In step (ii), we specify the functionality aspects if necessary. In particular, service capabilities like inputs or outputs are described in terms of concepts in a certain domain. Both sorts of attributes form the public interface.

The next step (iii), structural aspects are introduced by specifying one of the selection workflow patterns described previously, ensuring that only one concrete service is successfully executed. For example, the invocation of `Wind Info` contains two concrete operations, `Get Weather Report` and `Get Wind`. One incoming operation must be completed before executing the following service in the chain. Both concrete operations are executed but only one is considered by the workflow engine. Suppose that an `and-split` with `disc-join` selection pattern is chosen and `Get Wind` would terminate before `Get Weather Report`. In this case the latter is ignored in the workflow execution. Binding aspects are specified in (iv) where the appropriate transformation between different levels of abstraction are described. In this case, the binding concept establishes relationships between the inputs and outputs concepts of the integrated component with the service description specification (typically WSDL). For example, it is necessary to declaratively specify a mapping from an integrated component (`Wind Info`) to its concrete operations (`Get Wind and Get Weather Report`) by specifying network protocol, message format and, optionally, more elaborated transformations such as filters or logical views. Structural and binding attributes form the private interface. Finally, step (v) consists simply of registering (updating) the abstract operation as an available integrated component for future reuse.

4 Conclusion and Future Work

In this paper we have presented a methodology for developing web applications using three basic processes for service composition, namely Service Abstraction Process, Service Composition Process, and Translation Process. In order to better support service reuse, we introduced a model for service composition that implements such a methodology. We described the core elements in our model: workflow patterns and integrated components as building blocks for incrementally composing complex service compositions. Our ongoing work includes a toolset to support the integrated components definition and to explore service reuse during the composition process.

Acknowledgements

This work has been partially supported by the EU project IST-2001-37724, by the Spanish Ministry of Science and Technology project TIC-2003-09365-C02, and by Fundación Bancaixa-Castelló.

References

1. Aalst, W.M.P., Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases **14** (1), (2003) 5-51
2. Alonso, G. Casati, F., Harumi, K., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer-Verlag, Berlin Heidelberg (2004)
3. Granell, C., Poveda, J., Gould, M.: Incremental Composition of Geographic Web Services: an Emergency Management Context. In: Proc. of the 7th AGILE Conference on Geographic Information Science, University of Crete Press, Crete, Greece, pages 343-348 (2004)
4. Granell, C., Ramos, J.F.: An Object-oriented Approach to GI Web Service Composition. In: Proc. of the First DEXA Workshop on GIM 2004, Zaragoza, Spain, pages 835-839 (2004)
5. Grønmo, R., Solheim, I.: Towards Modeling Web Service Composition in UML. In: Proc. of The 2nd Intl. Workshop on Web Services: Modeling, Architecture and Infrastructure (WSMAI), Porto, Portugal, pages 72-86 (2004)
6. Melloul, L., Fox, A.: Reusable Functional Composition Patterns for Web Services. In: Proc. of the IEEE ICWS 2004, San Diego, California (2004)
7. Menascé, D.A.: QoS Issues in Web Services. IEEE Internet Computing **6** (6), (2004) 72-75
8. Milanovic, N., Malek, M.; Current Solutions for Web Service Composition. IEEE Internet Computing **8** (6)**,** (2004) 51-59
9. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. CUP (1999)
10. Sollazo, T. Handschuh, S., Staab, S., Frank, M.: Semantic Web Services Architecture – Evolving Web Service Standards towards the Semantic Web. In: Proc. of 15th International FLAIRS Conference. AAAI Press, pages 425-430 (2002)
11. Szyperski, C.: Component Software. Beyond Object-Oriented Programming. Addison-Wesley, New York (1998)
12. Szyperski, C.: Component Technology – What, Where, and How?. In: Proc. 25th International Conference on Software Engineering ICSE'03, pages 683-693 (2003)
13. Vinoski, S.: WS-Nonexistent Standards. IEEE Internet Computing **8** (6), (2004) 94-96.