Pattern Based Composition of Web Services for Symbolic Computations

Alexandru Cârstea[1], Georgiana Macariu[1], Dana Petcu[1], and Alexander Konovalov[2]

[1] Institute e-Austria Timişoara, România
science@ieat.ro
[2] University of St Andrews, St Andrews, Scotland
alexk@mcs.st-and.ac.uk

Abstract. The suitability of the BPEL workflow description language for the dynamic composition of Web services representing computational algebra systems is investigated. The prototype implementation of the system for dynamic generation of BPEL workflows and two examples demonstrating the benefits of our approach are described. One of important aspects of the design is that the composition is achieved using standard workflow patterns without any modification of the underlying computational algebra systems, provided they support the OpenMath format.

Keywords: dynamically generated workflows, service-oriented architecture, symbolic computing, workflow patterns.

1 Introduction

Complex problems can be solved using algorithms that combine multiple execution steps. Workflow technologies are often used nowadays to combine the results obtained by invoking black box software components available as Web services. Most of the classical composition examples are referring to static composition that is achieved at design time by specifying all the details of the composition. On the other hand, in the more technologically challenging case, of dynamic composition of Web services, the decision on which services have to be called in order to solve a particular problem is done at runtime.

Dynamic composition intends to make use of the tremendous potential offered by already existing Web services. Several practical problems prevent dynamic composition to be applicable as general solutions. First of all, the standard WSDL document describes the service interface but it doesn't offer any information regarding its functionality or its QoS. Another problem is the limited availability and transient nature of such services. While no current standard approaches for dynamic composition were developed, Section 2 is an overview of such techniques, general and applied in the area of symbolic computation.

The system that we recently proposed [1,2] focuses on exploiting the functionality offered by Computer Algebra Systems (CAS) wrapped as Web and Grid

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 126–135, 2008.

services. The previous results underlined the need and proved the capability to expose the functionality of various CASs through uniform interfaces exposed as Web services and demonstrated the ability to compose the functionality of those systems for problems that follow a certain pattern.

Complicated graphical interfaces currently allow creation and deployment of static workflows. While these solutions are extremely useful for specialized users, they are unusable in the context of common CAS interfaces due to their lack for the specific functionality required by CAS application developers. On the other hand, recent research results about Web services composition led to the identification of workflow patterns (Section 3 discusses some of them). The CAS users may benefit from the potential functionality provided by a software solution that allows combining Web service functionality using standard patterns.

The present paper focuses on extending the simple composition of symbolic computing Web services based on a given structure to a complex one based on an arbitrary workflow. The proposed solution for the construction and deployment of composed Web services in a dynamic fashion is planned to be available through the CAS' usual interface. General workflow patterns are helping the CAS user to describe the relationships and sequence of services calls. The resulted description is deployed and executed by components implemented using Java 5 SDK relaying on the ActiveBPEL [3] workflow engine and the PostgreSQL [4] database servers. The approach is described in Section 4, while some implementation details are pointed out in Section 5. The functionality of the system is presented using several examples in Section 6 and conclusions are outlined in Section 7.

2 Related Work

Specialized languages for describing Web service workflows are usually XML based languages because they are well suited for the automated machine processing. Graphical interfaces, e.g. ActiveBPEL Designer [3], can be used to create abstract or concrete workflows and assists the user in deploying the resulted workflow. Low level details such as the URL location of the partner services, may be explicitly provided by the user. Triana [5] can be used in conjunction with previously known UDDI registries to discover and compose Web service functionality. Platforms for managing composed services, such as EFlow [6] allow predefined static composition with a dynamic binding selection technique.

Dynamic composition approaches include AI planning mechanisms and ontology based composition. The set of services dynamically selected to solve a particular problem may change from one invocation to another. As a result, dynamic discovery mechanism must be used at runtime to decide which services should be invoked. The selection of services must meet requirements regarding the functionality and the QoS to be provided. In this respect, several general problems may appear [7]. The discovery problem, for example, raises two subproblems that need to be solved at the same time: obtaining a service description and obtaining the location of the service. Reliability constitutes also an issue since services may be occasionally unavailable.

In [8] it is noted that a generally accepted assumption is that each Web service can be specified by its preconditions and effects in the planning context. A similar assumption is also used in Polymorphic Process Model (PPM) [9]. A specialized language, DAML-S [10] has direct support for AI planning techniques. The state change produced by the execution of a Web service is specified through the precondition and effect properties of the service profile.

As described in [11], the semantic Web vision is to make Web resources accessible by content as well as by keywords. Web services play an important role in this scenario: users and software agents should be able to discover, compose, and invoke content using complex services. The main drawback of this approach is that specifying ontologies may become a very complicate task.

Symbolic computation services may be part of a computational infrastructure that can be used for solving complex problems. The analysis of the work conducted in the context of building symbolic computing services by projects such as MONET [12], GENSS [13] or MathBroker [14], has led us to the conclusion that dynamic discovery techniques implemented using AI techniques for Web services, in general, and for symbolic services, in particular, are not yet able to provide a wide-scale applicable solution. The discovery process in MONET uses the MSDL ontology language and the MPDL problem description language to retrieve the right mathematical services by interrogating modified UDDI registries. A similar agent based approach is also used in GENSS.

Our approach differs in several respects. First of all, it uses the functionality offered by remotely installed CASs as potential solvers of mathematically described problems. The current system aims to integrate the functionality of the functions implemented in remote CASs into the context of the user's CAS system. The discovery process uses as a main criterion of selection the functionality implemented by a certain service to manage a certain OpenMath call object. The OpenMath standard [15] ensures the interoperability between Web services that expose functionality of different CASs.

Previous results obtained in the context of workflow patterns [16] are used within the current approach to provide a higher level of abstraction. Thus, implementation details are hidden and the user can concentrate on the problem and not on low level details of implementation. The user can build arbitrary complex workflows using standard constructs (workflow patterns): the complex symbolic computation process is specified in terms of workflow patterns and not in a specific workflow composition language.

3 Workflow Patterns Background

Algorithmic solutions of complex problems are obtained through execution of atomic steps in a predefined order. The analysis of the algorithm implementations for different problems often led to the identification of higher level patterns. As a result of the research in the domain of Web services composition, specialized software components that are capable of executing workflows described using languages such as XLANG, WSFL and BPEL were created. The description of

these workflows requires low level details such as the address of the composed Web services, data conversion specification and fault handling.

Several patterns that apply to Web service composition were identified in [16] and they were further used to investigate the expressivity of several workflow languages and the support they offer for implementing various patterns [17]. A short overview of the most common workflow patterns is presented below.

A *sequence pattern* represents the sequential execution of two or more tasks. The dependency between certain steps may be purely functional, or a data dependency may exist between these tasks. When the nature of the problem to be solved permits it, several tasks may be executed in parallel as a *parallel split pattern* that describes a process fork. If the subprocesses reunite at a certain moment of the execution, that point is a join point and the parallel split is with synchronization. For this pattern we assume that every branch is executed only once. As a variation of this pattern, multiple instances without synchronization occurs when multiple instances of the same task must be executed.

A group of tasks may have to be executed only if a condition is met. Such behaviour may be expressed using *conditional patterns*. The *exclusive choice pattern* selects, amongst several branches, a branch that should be executed. Similarly, the *multichoice pattern*, allows several branches to be executed in parallel if the individual condition for each branch is met. One can potentially encounter more that one possible approach while solving a symbolic computation problem. Several solving techniques should be tested at the same time by concurrent processes and, as soon as the solution is obtained, the rest of the processes may be discarded. The *deferred choice pattern* expresses this functionality.

Often there are situations when the same action must be applied several times to various arguments. This behaviour is expressed as the *multiple instances with prior knowledge pattern* when the exact number of iterations is known and as *multiple instances without prior knowledge* when an external call is expected to end the loop execution.

Web service composition is achieved by issuing calls to partner Web services that may return a result or they may be intended to solely alter the general state of the system. The communication models used to interact with partner services were abstracted as several *conversational patterns*. We have chosen to implement two models of interaction. A common pattern, the *request/reply pattern*, allows a synchronous invoke of a partner service. The other one, the *one way invocation pattern* covers the situation when the sender only wants to transmit a message to the partner Web service and it does not expect a response message to be issued so the client may continue its execution.

More complex communication patterns can be established using the above described communication patterns. Asynchronous communication is useful when the required computation time is long. A compound pattern that we found particularly useful is a combination of two *request/reply patterns*, where the *"reply"* message is used only as a form of acknowledgment. In this situation the Web service client sends a request and receives the result at a later time as a call-back message. The client role is played by the workflow management engine that combines

partner Web services functionality. This behaviour allows a *non blocking asynchronous communication* between the workflow and the partner services. A common functionality is to enable the user to interrupt the execution of a running process. The pattern that specifies this behaviour is the *cancel pattern*.

4 An Architecture for Composing Symbolic Services

Symbolic computing often demands computational resources that are not available in the context of a local machine and not even in the context of super computers or specialized clusters. Moreover, the client may request functionality available with a general purpose CAS or it may require services from a CAS specialized on particular field. Integrating those systems into a broader distributed architecture offers the premise to use the best available software solution for a given problem. The solution we propose is based on a computational infrastructure that brings required hardware and software resources together, using HPC, Web and Grid related technologies.

CASs are the main tools for symbolic computing. To enable remote access to their functionality we have developed CAS Server components [1] that expose CAS functionality as Web services. For discovery and security reasons, the local registries store information about the CAS installed on the server, respectively the CAS functions that are available to be remotely invoked. More details about CAS Server components are given in [1].

Building on CAS Servers, we have implemented a system that is able to combine functionality of several CASs (Figure 1). A complex problem can now be solved by combining the results computed using different CASs and the computing power of a distributed architecture. Orchestration of multiple CAS Servers is a complex process that must offer solutions for discovering, invoking and storing results received from CAS Servers that were invoked. The key of success is the ability to express the solution in terms of workflow patterns. Using Web technologies the communication among CASs is simplified and standardized (a key in achieving this goal is the usage of the XML based OpenMath language).

Application specialists need not be aware of all the details required for the complete specification of the whole workflow using a specialized language. Instead, they only need to be able to combine several workflow patterns in order to describe a high level solution to their problem. The user-specified workflow can be automatically translated into a specialized workflow language, deployed and executed by a workflow management server. The blueprint of the client component that we have implemented can be used to enable this functionality within every CAS with a minimal effort. Thus, a solution for a certain problem can be described in terms of the supported workflow patterns. As we shall see in the examples section, our solution enables the GAP system [18] to combine workflow patterns and execute workflows that use the functionality of several other CASs installed on remote machines.

The description of the problem specified at the client level is submitted to a server that will manage the rest of the process. At the client side, the workflow

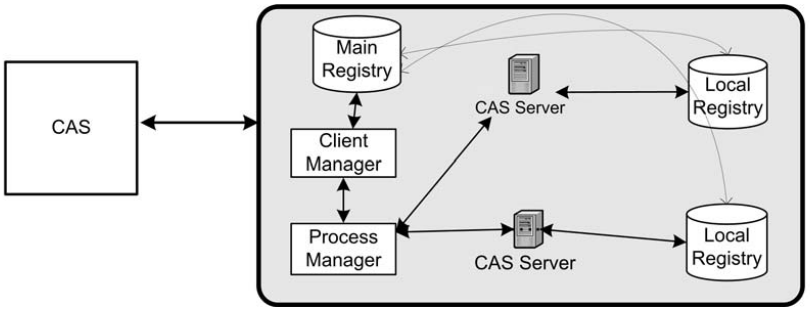

Fig. 1. CAS-wrapper service architecture

specified within the CAS is encoded using a XML language similar to BPEL. The main reason for using a XML intermediate language instead of a complete BPEL description is the significant larger size of the complete specified BPEL workflow. Additionally, this approach enable clients with few computational and communication available resources, e.g. PDAs and mobile phones, to access the system. The drawback of this approach is the additional server load needed to convert the XML format to the BPEL format.

In a distributed system one cannot correctly predict the status of the computing infrastructure available to be used for solving the problem. Our system enables the user to combine the functionality offered by CASs installed on the CAS Servers registered to the system; the user is able to specify the CASs to be used, but the particular CAS Server that is invoked at runtime is selected automatically by the system, based on several relevant criteria. The most important criterion is the functionality provided by a particular CAS Server. Another criterion is the current load of a hardware resource. The current paper does not focus on finding the best selection or scheduling algorithm to be used. The system will be enriched with a load balancing facility in the near future.

Several changes had to be implemented on the simple system presented in [2] to support the functionality described above. The client manager component is now responsible not only for receiving new workflows and for providing clients access to the result of their computation, but also for translating the XML workflow representation received from the client to the corresponding BPEL workflow format, to deploy the workflow into the ActiveBPEL engine and to launch the execution of the process.

5 Implementation Details

Using the system presented in this paper, the client is able to specify workflows by composing standard workflow patterns. This functionality is based on the implementations of several workflow patterns. We have chosen BPEL as a workflow description language due to its better capabilities comparatively to its competitor languages, demonstrated [19]. This system implements workflow

patterns using BPEL predefined activities and additional constructs. While the activities represent the structure of the workflow, additional technologies such as XSD and WSDL are used to support data dependencies amongst activities, to evaluate conditions and to identify the partner services.

Several patterns, such as the sequence pattern, have direct correspondence with existing BPEL activities, but most of the patterns have to be implemented by complex constructions. Using the Java API offered by the ActiveBPEL engine we generate constructions similar to those described in [17]. Patterns that can be implemented with minimal efforts are the sequence pattern and the parallel/split pattern because of the direct correspondence for these patterns in BPEL through the sequence and flow BPEL activities. We were also able to implement patterns like exclusive choice, or multiple choices with and without synchronization.

Conversational patterns cannot be implemented straightforwardly because they require adding the corresponding invocation task, input and output variables and links with partner WSDL documents to the resulting BPEL document. The lack of prior knowledge about the structure of the new workflow imposes that these details are generated at deployment time. Predefined structure of Web services' interfaces and a standard encoding format for the data representation, namely OpenMath, makes composing these services possible. In the context of arbitrary Web services, implementing conversation patterns would be impossible without additional semantic information being available.

Encoding data using OpenMath is legitimate because the content of the messages is intended to be understood and used in the context of a CAS. The workflow engine does not have the ability to manage OpenMath objects and is not expected to understand the content of the data exchanged among partners. As a side effect, the current version of the system has certain limitations regarding the way the conditions described for conditional patterns and repetitive patterns must be specified. For example, an OpenMath object that does not encode a number cannot be used to specify a condition.

The process specified at the client level is translated by the Client Manager component into a BPEL workflow. The main part of the resulted BPEL document is the corresponding translation of the workflow described at client side. Starting the workflow can be done only by invoking the composed service that results after deploying the workflow, therefore an additional receive activity had to be added. Results obtained after the execution of the workflow are sent to a Web service responsible for storing the results through an additional call. Because we want to avoid the computational expense of deploying the same workflow several times, we allow the user to access already deployed workflows.

6 Examples

The previously implemented approach [2] offers the ability to execute simple scenarios. An example that was used to demonstrate its functionality was the ability to compute the value of Gcd(Bernoulli(1000), Bernoulli(1200)) using remote machines and two different CASs: GAP and KANT. The Gcd() was computed using

a KANT system by combining the Bernoulli results obtained from two separate instances of GAP. We used this example as a starting point for demonstrating the capabilities of the system. The main functional enhancement of the system described here is that it permits execution of workflows that are not bound to a two level invocation scheme. The corresponding GAP code that would allow obtaining the same result as the previous system is:

```
startWorkflow();
    startSequence();
        startParallel();
            v1:=invoke("KANT",Bernoulli(1000));
            v2:=invoke("KANT",Bernoulli(2000));
        endParallel();
        invoke("GAP",gcd(v1,v2));
    endSequence();
endWorkflow();
```

The above code is translated at the client level into a simplified BPEL like format and it is submitted to a server. The server will translate the workflow into a regular BPEL workflow and will manage the execution. At a later time, the user may access the computed result based on the identifier that it is received when submitting the workflow.

The next example describes the "ring" workflow. Imagine a "ring" of services, where each service accepts request from its "left" neighbour (for example, an integer N), performs an action (for example, N:=N +1) and sends the new value of N to its "right" neighbour. The test is started with the initial value N=0 and will be terminated by that service on which the parameter N will reach the prescribed upper bound.

Below we demonstrate the pseudocode describing a generic ring workflow for two services that can be straightforwardly extended for arbitrary number of services (running the same or various CASs) to combine them in a ring:

```
startWorkflow();
    c:=setCondition("N<100");
    startWhile(c);
        startSequence();
            v:=invoke("GAP","Int(N+1)");
            c:= setCondition("N<100");
            startMultiChoice();
                startBranch(c)
                    v:=invoke("KANT","EvalString(N+1)");
                endBranch();
            endMultiChoice();
            c:= setCondition("N<100");
        endSequence();
    endWhile();
endWorkflow();
```

By implementing the "ring workflow" example we demonstrate that the system can be used to implement complicated workflows. For example, in the workflow arising from the orbit enumeration algorithm [20] we can combine three various kinds of services:

1. *job server*, sending procedure calls to appropriate *image service*.
2. *image service* for computing the image of the point (may be more than one, each sending procedure call to appropriate *orbit service*).
3. *orbit service* for storing the orbit (may use hash tables, may be more than one, each maintaining part of the table and sending procedure call, if necessary, to the *job server*).

7 Remarks and Future Work

This paper is focused on using common workflow patterns as high level components to express and execute dynamic generated workflows for symbolic computing. The software solution that we provide has several additional benefits.

The user specifies the workflow to be executed and then submits it to a server. The result can be obtained at a later time without having to maintain the connection with the server. Usually, the connection provided between servers is better that the connection of a client to a server. Managing the workwflow at the server level may allow the client to obtain the result faster than managing the workflow at the client side. The server may also provide load balancing and failure management by using a specialized workflow engine. Since the description of the workflow is done in XML format, the proposed system can also be used not only by a CAS client, but also by any system that is able to properly formulate the request and to submit it through the Web service interface of the execution engine provided by the system.

The system may be extended in future versions by offering additional support for several workflow patterns that were not implemented yet and will emerge as needed in the intensive testing phase and by adding better selection policies of CAS Servers. An important issue that we want to tackle in a future version is to enable support for Grid services since their specific interface prevents the current system to compose the functionality of these services.

Acknowledgements. This research is partially supported by EU FP6 grant RII3-CT-2005-026133 SCIEnce: Symbolic Computing Infrastructure for Europe.

References

1. Cârstea, A., Frîncu, M., Konovalov, A., Macariu, G., Petcu, D.: On Service-oriented Symbolic Computing. In: Wyrzykowski, R. (ed.) PPAM 2007. LNCS, vol. 4967. Springer, Berlin (inprint, 2008)
2. Cârstea, A., Macariu, G., Frîncu, M., Petcu, D.: Composing Web-based Mathematical Services. In: Negru, V., et al. (eds.) SYNASC 2007, pp. 327–334. IEEE Computer Society Press, Los Alamitos (2007)

3. ActiveBPEL, http://www.active-endpoints.com/active-bpel-engine-overview.htm
4. PostgreSQL, http://www.postgresql.org/
5. Majithia, S., Shields, M., Taylor, I., Wang, I.: Triana: a Graphical Web service Composition and Execution Toolkit. In: ICWS 2004, pp. 514–521. IEEE Computer Society, Washington (2004)
6. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.C.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)
7. Solomon, A.: Distributed Computing for Conglomerate Mathematical Systems. In: Joswig, M., Takayama, N. (eds.) Algebra, Geometry and Software System, pp. 309–325. Springer, Berlin (2003)
8. Rao, J., Su, X.: A Survey of Automated Web Service Composition Methods. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005)
9. Schuster, H., Georgakopoulos, D., Cichocki, A., Baker, D.: Modeling and Composing Service-based and Reference Process-based Multi-enterprise Processes. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 247–263. Springer, Heidelberg (2000)
10. Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D.L., Mcllraith, S.A., Narayanan, S., et al.: DAML-S: Semantic Markup for Web Services. In: Cruz, I.F., et al. (eds.) The Emerging Semantic Web ISWC 2002. IOS Press, Amsterdam (2002)
11. Milanovic, N., Malek, M.: Current Solutions for Web Service Composition. IEEE Internet Computing 8(6), 51–59 (2004)
12. Aird, M.L., Medina, W.B., Padget, J.: MONET - Service Discovery and Composition for Mathematical Problems. In: IEEE/ACM CCGrid 2003, pp. 678–685. IEEE Computer Society, Los Alamitos (2003)
13. Grid-Enabled Numerical and Symbolic Services, http://genss.cs.bath.ac.uk/index.htm
14. Baraka, R., Schreiner, W.: Querying Registry-published Mathematical Web Services. In: Wagner, R., Ma, J., Durresi, A. (eds.) AINA 2006, pp. 767–772. IEEE Computer Society, Los Alamitos (2006)
15. OpenMath, http://www.openmath.org/
16. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow Control-flow Patterns: A revised view. BPM Center Report BPM-06-22 (2006)
17. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Analysis of Web Services Composition Languages: The case of BPEL4WS. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 200–215. Springer, Heidelberg (2003)
18. The GAP Group, GAP - Groups, Algorithms, and Programming, Version 4.4.10 (2007) http://www.gap-system.org
19. Kiepuszewski, B.: Expressiveness and Suitability of Languages for Control Flow Modeling in Workflows. PhD thesis, Brisbane, Australia (2003)
20. Lübeck, F., Neunhöffer, M.: Enumerating Large Orbits and Direct Condensation. Experiment. Math. 10(2), 197–205 (2001)