Requirements for a Workflow System for Grid Service Composition

Niels Joncheere, Wim Vanderperren, and Ragnhild Van Der Straeten

System and Software Engineering Lab (SSEL)
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
{njonchee,wvdperre,rvdstrae}@vub.ac.be

Abstract. In this position paper, we propose a new generation workflow system for grid services. We observe that grid computing has become an increasingly important application domain in computer science. Grid services — a new technology based on web services — are expected to become the de facto standard for grid computing. Similar to web services, an effective mechanism is needed for the composition of grid services. Existing technologies, however, have a number of important drawbacks: they have limited or no support for modularization of crosscutting concerns, for dynamic workflow adaptation, and for high-performance computing. We propose a new generation workflow system that is tailored specifically for grid services, and that tackles these problems, among others.

Keywords. Aspect-oriented software development, grid services, workflow languages

1 Introduction

Over the last years, *grid computing* has become an increasingly important research domain within computer science. "The Grid" can be described as *a service for sharing computing power and data storage capacity over the Internet*.[1] The specific problem that lies at the heart of this technology is *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations* [1]: the large scope of many current scientific problems makes it increasingly difficult to solve them using only one computer system, and forces the use of a distributed solution. By creating a virtual organization of different resources, which typically don't belong to the same owner but are connected through the Internet, it is possible to address these problems. A recent challenge is the use of grid technology beyond scientific applications, more specifically in the production and design of products in industrial environments. Such design tasks typically require the use of multiple complex simulation- and optimization tools, which are used as part of a design process workflow.

[1] http://gridcafe.web.cern.ch/gridcafe/

Since 2002, a standardization effort for grid computing has been active under the form of the Open Grid Services Architecture (OGSA).[2] This initiative aims to promote the acceptance and application of grid technologies through standardization. Its main task is the harmonization of academic activities concerning the Grid, with *web services* [2], a technology which also has a lot of industry support. Web services are applications that are accessible through the Internet and which use SOAP/XML for the transmission of information and WSDL/UDDI for the description and discovery of other web services. The OGSA standardization work has led to the development of *grid services* [3], which are actually a subclass of web services, with additional properties relevant for grid computing. It is expected that grid services will soon become the de facto standard for grid computing.

Although the services themselves have been standardized, composing grid services is still an open issue. Grid services are currently typically composed by manually writing the necessary glue-code in programming languages such as C and Java. In the web services world, however, it has been identified that a composition of web services is more naturally captured by dedicated workflow languages than by general-purpose programming languages. Languages such as BPEL4WS [4] and WS-CDL [5] have already been well accepted in the web services community. Because grid services are a kind of web services, it is in principle possible to recuperate these workflow languages for grid services. However, we identify several problems with current practice web service workflow languages:

- Most workflow languages do not have a clearly defined semantics [6].
- Most languages are not suitable for high-performance computing.
- There is insufficient support for the modularization of *crosscutting concerns* [7].
- There is insufficient or no support for dynamic adaptation of workflows.

The goal of this position paper is to identify the main limitations of current approaches for grid service composition, and to specify the requirements of a workflow system that is specifically tailored for grid service composition, and which therefore addresses these limitations, among others.

The outline of the paper is as follows. In Section 2, we analyze current approaches for grid service composition and identify their main limitations. In Section 3, we specify the requirements for a workflow system for grid services based on these limitations. In Section 4, we illustrate the applicability of our approach by describing the context in which it would typically be used. In Section 5, we present related work, and in Section 6, we state our conclusions.

2 Limitations of Current Approaches

Several workflow approaches for grid computing currently exist [8–13]. These approaches, however, use their own service and communication standards, which

[2] `http://www.globus.org/ogsa/`

makes them incompatible with each other. Furthermore, most of them are not suited for the new grid services standard. Therefore, composing grid services is currently typically handled by manually programming glue-code. Existing workflow systems for web services can technically be used for composing grid services, as grid services are a special kind of web services. There are, however, several problems with current workflow systems for web services which make them less suited for the grid services context.

Currently, BPEL4WS is well supported and widely accepted as the de facto standard for web service composition. We therefore limit our discussion of limitations to BPEL4WS; most of the limitations are also applicable to the other, less frequently used approaches. Note that our approach itself is not based on BPEL4WS; we merely discuss it here because it is representative of the state of the art in workflow languages.

2.1 Semantics

van der Aalst [6] observes that most workflow languages for web services do not have a clearly defined semantics. BPEL4WS is a combination of WSFL [14] and XLANG [15], which are in their turn based on other, earlier languages. A large amount of the functionality of the original languages has ended up in the new language. This makes BPEL4WS expressive at the price of being complex.

2.2 High-performance Computing

Another important topic regarding workflow languages for grid services is the support for the specific requirements that are typically of importance with high-performance computing. In such an environment, it is common that large amounts of data need to be transferred from one step in a workflow to another. Special attention should be directed to how this happens: it would, for example, be unacceptable if all these data were transferred to a central workflow coordinator before being transferred to a next step. This is, however, current practice in BPEL4WS workflow systems.

2.3 Separation of Concerns

BPEL4WS does not have sufficient support for separation of concerns [16]. It is very difficult to modularize BPEL4WS processes in an effective way, because each process must be specified in a single XML file. Furthermore, it is not possible to specify sub-processes separately. Complex processes thus give rise to large XML files, which can become difficult to understand and maintain. A workaround for this problem is to expose sub-processes as separate services, but this is not always desirable, as it introduces additional overhead and scoping problems.

Even if BPEL4WS would support sub-processes, one would still not be able to modularize every concern successfully. Kiczales *et al.* [7] recognize that some concerns of an application cannot be modularized using existing software development methods: the implementation of these concerns is completely spread out

over the different modules of the system. The same logic is repeated among different modules, resulting in code duplication. This code duplication makes it very difficult to add, modify or remove these concerns [17]. Examples of such *crosscutting concerns* are security concerns such as access control and confidentiality [18], debugging concerns such as logging [19] and timing contract validation [20], and business rules such as billing [21].

The goal of *aspect-oriented software development* (AOSD) is to achieve a better separation of concerns, by allowing crosscutting concerns to be specified in separate modules called *aspects*, so that adding, modifying or removing these concerns does not require changes to the rest of the system. As the crosscutting concerns we mentioned above are encountered frequently in BPEL4WS workflows [22–25], a solution for better modularization based on aspect-oriented techniques is necessary.

2.4 Dynamism

BPEL4WS does not support altering the workflow specification while it is running. The only exception is altering the concrete partner bindings (web services) when the additional standard WS-Addressing [26] is employed. In a grid services context, where long-running computations are the norm, dynamic workflow adaptation is essential in order to manage changing requirements.

3 A New Generation Workflow System

The goal of our approach is to offer a new generation workflow system, which is specifically tailored for grid service composition, and which therefore addresses the problems that were identified above, among others. This section specifies the requirements for this workflow system. These requirements can be divided into six properties, namely workflow, AOSD, dynamism, modularity, high-performance computing, and semantics. Each of these properties is discussed in further detail below.

3.1 Workflow

Every workflow language needs to define the basic activities that it supports, and how these activities can be ordered. Concerning the basic activities, we support invoking grid services (both synchronously and asynchronously), and assigning and retrieving variables.

Concerning the ordering of activities, existing literature is useful when deciding which kinds of orderings must be supported. For example, van der Aalst *et al.* [27] have identified a number of recurring workflow patterns, from elementary to complex, based on an extensive study of existing workflow languages. These patterns can be divided into six categories: basic control patterns, advanced branching- and synchronization patterns, structural patterns, patterns involving

multiple instances, state-based patterns, and cancellation patterns. Our workflow language naturally supports the basic control patterns (such as sequence and exclusive choice), but regarding the more complex patterns, we need to weigh expressivity against clarity.

3.2 AOSD

Just like with regular software, some concerns of a grid service composition (such as billing and logging) cannot be modularized using current technologies: they end up scattered across the composition, and tangled with one another. This makes it difficult to add, modify or remove these concerns. In order to avoid such problems, one of the main properties of our workflow system is that it supports AOSD in order to allow the modularization of crosscutting concerns.

We propose a joinpoint model that allows advices to be executed before, around and after each basic workflow activity (e.g. service invocations and variable assignments). Pointcuts are expressed in a language based on logic programming, which allows selecting joinpoints based on the names and types of the corresponding workflow activities. Advices are expressed in the basic workflow language.

Because workflows involving grid services typically run for a long time, and on expensive infrastructure, it is important that the chance of encountering unexpected behavior is minimized. Therefore, we require that all properties of aspect-oriented interactions (such as the order in which multiple advices, which are applicable on the same joinpoint, are applied) are specified in advance.

3.3 Dynamism

Grid workflows often take a lot of time to complete, because of the complicated calculations and the large amounts of data that are involved. Additionally, they run on complicated, expensive infrastructure, which may be used on a pay-per-use basis. This makes it prohibitively expensive if workflows do not behave as expected, and have to be restarted.

This problem is our motivation for requiring that all properties of aspect-oriented interactions are specified in advance. However, this does not solve the problem completely: suppose that a certain business unit of a company is awaiting the results of a grid workflow that is taking more time to complete than expected. In such a case, it could be useful if the workflow could be modified while it is running in order to get it to finish faster (e.g. by removing certain parts of the workflow that are not considered essential to obtain the results needed by the business unit).

As another example, consider the case where an error is discovered near the end of a workflow that has already performed a lot of useful computations. In this case, correcting part of the workflow while it is running would certainly be preferable to terminating it and restarting it after the workflow is corrected.

Therefore, our workflow system supports dynamic workflow adaptation, i.e. it is possible to modify workflows while they are being executed. We allow pieces

of workflow to be added, replaced or removed in any place where the control flow has not yet passed at the time of the modification. We aim to prevent introducing specific language constructs for this purpose: a piece of workflow should not need to know that part of it might be adapted during its execution, or that it might be used to replace another piece of workflow.

The dynamism we discussed above concerns the basic workflow description. However, dynamism can be useful with respect to AOSD, too: several aspect-oriented programming (AOP) languages [28, 29] already support dynamic enabling and disabling of aspects in order to facilitate adapting to concrete situations. Therefore, our workflow system supports such *dynamic AOP*, too. Although dynamic AOP once had a reputation of introducing high performance overhead, recent work [30] has shown that it is possible to implement dynamic AOP efficiently using techniques such as just-in-time compilation and caching.

3.4 Modularity

Because of our system's support for AOSD, it facilitates modularizing crosscutting concerns. Using current workflow languages, however, it is not always possible to effectively modularize even the basic workflow. For example, a BPEL4WS process is always a single monolithic specification, which makes it impossible to reuse parts of a process elsewhere (unless these parts are modeled as separate web services, which introduces a large amount of overhead).

Therefore, we support sub-processes by allowing parts of a workflow to be specified in separate modules. These modules can then evolve independently from the main workflow, and can be reused in other workflows. It is clear that such an approach is an improvement on a number of current approaches.

3.5 High-performance Computing

In traditional web services applications, the messages that are exchanged between services are typically very small. In grid computing, on the other hand, it is common that large streams of data need to be transferred. Therefore, requiring that all data pass through a central workflow coordinator — as is the case with conventional workflow languages such as BPEL4WS — is unacceptable, as it would require much more network capacity than is actually necessary. We therefore aim to remedy this problem by providing a distributed workflow coordinator that makes sure large data streams are routed directly to the next step in the workflow.

In order to provide this functionality, we introduce a language construct that allows specifying which data streams may be large and may thus require more efficient modes of network transport. On the other hand, the workflow engine may decide which streams of data are large, and handle them accordingly, if such information is not specified.

3.6 Semantics

It has been argued that most current workflow languages do not have a clearly defined semantics [6]. Among others, this hampers compatibility between different engines for a same workflow language. Therefore, we aim to define a formal operational semantics for our workflow language.

4 Applicability

Existing work on grid computing mostly focuses on scientific applications, such as bio-informatics or high-energy physics. Our approach, however, is aimed at design and production in industrial settings — an increasingly important application domain for grid services.

A lot of design activities in such industrial setting traditionally require prototypes to be built and tested in order to obtain important information on the design (e.g. crashing an automobile into a wall in order to discover safety information, placing a model of a plane in a wind tunnel in order to discover aerodynamics information, etc.). Innovation in computer science has made it possible to simulate this kind of tests. Because this kind of virtual prototyping requires a lot of computing power and involves a lot of data, such problems cannot be solved easily using only one computer system. Using grid computing, however, these problems can be solved effectively.

The industrial partner for our research currently sells such virtual prototyping grid software to clients that wish to test their products. These clients then use their own grid infrastructure to run the simulations. This limits the applicability of the industrial partner's products, as the required infrastructure's cost may be prohibitively high.

Our workflow system would, however, enable a completely new business model, in which clients use the infrastructure of the industrial partner on a customizable basis. In such a scheme, clients would not be forced to invest in specific infrastructure any more, and the industrial partner would not be forced to spend time on offering support for clients' local software.

The contracts of the industrial partner with its clients may vary a lot: one client may require other grid services than another, the service level agreements may differ, or different billing schemes (pay-per-use, flat-fee, etc.) may be used. Our workflow system would facilitate enforcing such different contracts, as these could be modularized as aspects.

Of course, the applicability of our approach is not limited to the virtual prototyping case presented above: the properties of our system, such as support for AOSD and dynamic workflow adaptation, are also useful in the traditional scientific application domain of grid computing, and even in the application domain for web services.

5 Related Work

An aspect-oriented extension to BPEL4WS — AO4BPEL [23] — has been proposed. This extension supports dynamic adaptation of aspects. However, because it is an extension to BPEL4WS, it inherits the deficiencies identified in this paper, such as limited support for modularization (of non-crosscutting concerns) and high-performance computing. Another approach that recuperates aspect-oriented ideas in the domain of web services is the Web Services Management Layer [25]. The WSML modularizes redirection, advanced selection policies, and management concerns such as caching and billing using aspects. The WSML does, however, not directly support web service composition and relies on BPEL4WS for that end. As such, it also inherits the disadvantages discussed in this paper.

Currently, grid services are mostly composed manually, by writing programs in traditional programming languages (such as C and Java), which use libraries such as the ones provided by the Globus Toolkit[3] to interact with concrete grid services. This situation obviously has a lot of drawbacks, as these languages do not support dynamic adaptation of the composition, or modularization of crosscutting concerns. Recently, however, a number of approaches have been proposed that aim to remedy these problems.

GridNexus [31] is a graphical system for creating and executing scientific workflows in a grid environment. A GUI allows developers to specify processes by creating directed acyclic graphs whose nodes perform simple computing tasks, or invoke grid services. Processes can be saved as composites, which can then be reused in other processes. Visual process specifications are represented by scripts written in a language called JXPL, which can be executed by an appropriate engine. By using such scripts, the user interface is separated from workflow execution. Although this approach is a serious improvement on manual grid service composition, it is targeted mainly at scientific grid applications, and not at industrial applications. It does not support dynamic workflow adaptation nor advanced separation of concerns.

Kepler [32] is a graphical workflow system similar to GridNexus (both approaches even use the same GUI technology). Like GridNexus, processes are directed acyclic graphs. The most important difference is that Kepler does not translate diagrams to scripts in order to execute workflows: workflows are executed by the GUI, thus increasing coupling between process definition and execution. Kepler is also aimed at scientific applications, and does not support dynamic workflow adaptation nor advanced separation of concerns.

6 Conclusions

In this position paper, we observe that, although the application domain for grid services is rapidly expanding, current approaches for grid service composition

[3] `http://www.globus.org/toolkit/`

have a number of disadvantages that limit their applicability, and thus hamper the acceptance of grid services in industrial settings. The most important disadvantages are insufficient support for AOSD, for dynamic workflow adaptation, and for high-performance computing.

Therefore, we propose a new generation workflow system, which is specifically tailored for grid service composition, and thus tackles these problems, among others. We present the requirements for our workflow system, and illustrate its applicability. Our future work will be directed at performing a first iteration on the design and implementation of our system.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications **15**(3) (2001) 200–222
2. Alonso, G., Casati, F., Kuno, H., Machiraju, V., eds.: Web Services: Concepts, Architectures and Applications. Springer-Verlag, Heidelberg, Germany (2004)
3. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002) `http://www.globus.org/alliance/publications/papers/ogsa.pdf`.
4. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services version 1.1 (2003) `http://www.ibm.com/developerworks/library/ws-bpel/`.
5. Kavantzas, N., Burdett, D., Ritzinger, G.: Web Services Choreography Description Language version 1.0. W3C Working Draft 27 April 2004, World Wide Web Consortium (2004) `http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/`.
6. van der Aalst, W.M.P.: Don't go with the flow: Web services composition standards exposed. IEEE Intelligent Systems **18**(1) (2003) 72–76
7. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. Technical Report SPL97-008 P9710042, Xerox PARC (1997)
8. Bhatia, D., Burzevski, V., Camuseva, M., Fox, G., Furmanski, W., Premchandran, G.: WebFlow — a visual programming paradigm for web/Java based coarse grain distributed computing. Concurrency — Practice and Experience **9**(6) (1997) 555–577
9. Basney, J., Livny, M.: Deploying a high throughput computing cluster. In Buyya, R., ed.: High Performance Cluster Computing: Architectures and Systems, Volume 1. Prentice Hall (1999)
10. Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., Darlington, J.: Optimisation of component-based applications within a grid environment. In: Proceedings of the 14th International Conference on High Performance Computing and Communications (SC 2001), Denver, CO, USA (2001)
11. Romberg, M.: The UNICORE grid infrastructure. Scientific Programming, Special Issue on Grid Computing **10**(2) (2002) 149–157
12. Lorch, M., Kafura, D.: Symphony — a Java-based composition and manipulation framework for computational grids. In: Proceedings of the 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany (2002) 136–143

13. Gannon, D., Bramley, R., Fox, G., Smallen, S., Rossi, A., Ananthakrishnan, R., Bertrand, F., Chiu, K., Farrellee, M., Govindaraju, M., Krishnan, S., Ramakrishnan, L., Simmhan, Y., Slominski, A., Ma, Y., Olariu, C., Rey-Cenvaz, N.: Programming the grid: Distributed software components, P2P and grid web services for scientific applications. Cluster Computing **5**(3) (2002) 325–336
14. Leymann, F.: Web Services Flow Language (WSFL 1.0). IBM (2001)
15. Thatte, S.: XLANG — web services for business process design. Microsoft (2001) `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`.
16. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Comm. ACM **15**(12) (1972) 1053–1058
17. Elrad, T., Filman, R.E., Bader, A.: Aspect-oriented programming. Comm. ACM **44**(10) (2001) 29–32
18. De Win, B., Joosen, W., Piessens, F.: Developing secure applications through aspect-oriented programming. In Filman, R.E., Elrad, T., Clarke, S., Akşit, M., eds.: Aspect-Oriented Software Development. Addison-Wesley, Boston (2005) 633–650
19. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ. In Knudsen, J.L., ed.: Proc. ECOOP 2001, LNCS 2072, Berlin, Springer-Verlag (2001) 327–353
20. Vanderperren, W., Suvée, D., Jonckers, V.: Combining AOSD and CBSD in PacoSuite through invasive composition adapters and JAsCo. In: Proceedings of Net.ObjectDays 2003, Erfurt, Germany (2003) 36–50
21. D'Hondt, M., Jonckers, V.: Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. In Lieberherr, K., ed.: Proc. 3rd Int' Conf. on Aspect-Oriented Software Development (AOSD-2004), ACM Press (2004) 132–140
22. Arsanjani, A., Hailpern, B., Martin, J., Tarr, P.: Web services: Promises and compromises. Queue **1**(1) (2003) 48–58
23. Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In Zhang, L.J., ed.: Proceedings of the 2nd European Conference on Web Services (ECOWS 2004), Erfurt, Germany, Springer-Verlag (2004) 168–182
24. Cottenier, T., Elrad, T.: Dynamic and decentralized service composition with Contextual Aspect-Sensitive Services. In: Proceedings of the 1st International Conference on Web Information Systems and Technologies (WEBIST 2005), Miami, FL, USA (2005)
25. Verheecke, B., Vanderperren, W., Jonckers, V.: Unraveling crosscutting concerns in web services middleware. IEEE Software **23**(1) (2006) 42–50
26. Box, D., Christensen, E., Curbera, F., Ferguson, D., Frey, J., Hadley, M., Kaler, C., Langworthy, D., Leymann, F., Lovering, B., Lucco, S., Millet, S., Mukhi, N., Nottingham, M., Orchard, D., Shewchuk, J., Sindambiwe, E., Storey, T., Weerawarana, S., Winkler, S.: Web Services Addressing (WS-Addressing). W3C Member Submission 10 August 2004, World Wide Web Consortium (2004) `http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/`.
27. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(3) (2003) 5–51
28. Popovici, A., Gross, T., Alonso, G.: Dynamic weaving for aspect-oriented programming. In Kiczales, G., ed.: Proc. 1st Int' Conf. on Aspect-Oriented Software Development (AOSD-2002), ACM Press (2002) 141–147
29. Suvée, D., Vanderperren, W.: JAsCo: An aspect-oriented approach tailored for component based software development. In Akşit, M., ed.: Proc. 2nd Int' Conf. on Aspect-Oriented Software Development (AOSD-2003), ACM Press (2003) 21–29

30. Vanderperren, W., Suvée, D.: Optimizing JAsCo dynamic AOP through HotSwap and Jutta. In Filman, R., Haupt, M., Mehner, K., Mezini, M., eds.: DAW: Dynamic Aspects Workshop. (2004) 120–134
31. Brown, J.L., Ferner, C.S., Hudson, T.C., Stapleton, A.E., Vetter, R.J., Carland, T., Martin, A., Martin, J., Rawls, A., Shipman, W.J., Wood, M.: GridNexus: A grid services scientific workflow system. International Journal of Computer & Information Science **6**(2) (2005) 72–82
32. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., Mock, S.: Kepler: An extensible system for design and execution of scientific workflows. In: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), Santorini, Greece (2004)