Requirements for an Aspect-Oriented Workflow System for Grid Services

Niels Joncheere
System and Software Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
njonchee@vub.ac.be

Wim Vanderperren
System and Software Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
wvdperre@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. We evaluate the impact of our proposed workflow system on several software engineering properties, in particular on comprehensibility, predictability and evolvability.

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* [2]. The specific problem that lies at the heart of this technology is *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations* [16]: 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.

Submitted to the AOSD 2006 Workshop on Software Engineering Properties of Languages and Aspect Technologies (SPLAT 2006)

Since 2002, a standardization effort for grid computing has been active under the form of the Open Grid Services Architecture (OGSA) [4]. 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* [5], 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* [15], 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 instead of a general-purpose programming language. Languages such as BPEL4WS [7] and WS-CDL [19] 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 [29].
- Most languages are not suitable for high-performance computing.
- There is insufficient support for the modularization of *crosscutting concerns* [21].

- There is insufficient or no support for dynamic adaptation of workflows.

In this position paper, we analyze the current situation of grid service composition and identify the main limitations. Based on these limitations, we propose the requirements for a workflow system for grid services. We discuss how the comprehensibility, predictability and evolvability are affected when such a system is available. Finally, we present related work and state our conclusions.

2. LIMITATIONS OF CURRENT APPROACHES

Several workflow approaches for grid computing currently exist [9, 8, 17, 26, 23, 18]. However, they use their own service and communication standards, which makes them incompatible with each other. Furthermore, none of them is suited for the new grid services standard. For this end, 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. However, there are 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 this end, we will limit our discussion of limitations to the BPEL4WS approach. Most of the limitations are also applicable to the other, less frequently used approaches.

2.1 Semantics

van der Aalst [29] observes that most workflow languages for web services do not have a clearly defined semantics. BPEL4WS is a combination of WSFL [22] and XLANG [28], which are on their turn combinations of other, earlier languages. A large amount of the functionality of the original languages has ended up in the new language. Because of this and other reasons, the language is not very clear.

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 [24]. It is very difficult to modularize BPEL4WS processes in an effective way because each process must be specified in a single XML file. It is not possible to specify sub-processes straightforwardly. Complex processes thus give rise to large XML files, which can become difficult to understand and maintain. A workaround for this problem is to define 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, some concerns still could not be modularized successfully. Examples of these concerns are security concerns such as access control and confidentiality [13], debugging concerns such as logging [20] and timing contract validation [31], and business rules such as billing [14]. These concerns are encountered frequently in typical BPEL4WS workflows, and thus a solution for better modularization based on aspect-oriented techniques [21] 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 [10] is employed. In a grid services context, where long-running computations are the norm, dynamic adaptation is essential in order to manage changed requirements.

3. A NEW GENERATION WORKFLOW SYSTEM

This section specifies the requirements of our new generation workflow system. These can be divided into six properties, namely workflow, AOSD, dynamism, modularity, high-performance computing, and semantics. Each of these properties will be 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. We may select other activities at a later time.

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.* [30] 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, variable assignments, etc.). Pointcuts are expressed in a language that 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 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 already has 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 some 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 AOP languages [25, 27] already support dynamic enabling and disabling of aspects in order to facilitate adapting to concrete situations. Therefore, our workflow system supports dynamic AOSD, too.

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 one 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 allow parts of a workflow to be specified in separate modules, which 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 passes 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 [29]. 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. IMPACT ON SOFTWARE ENGINEERING PROPERTIES

This section evaluates the impact of our approach on a number of important software engineering properties, such as comprehensibility, predictability, and evolvability.

4.1 Comprehensibility

Due to the better support for modularization of both crosscutting and non-crosscutting concerns, we claim that the comprehensibility is significantly improved. By isolating each concern in a separate module, it becomes easier to comprehend and maintain the system.

4.2 Predictability

Our system actually aims for maximum reliability and predictability. For this end, all properties have to be explicitly defined up-front. For instance, there are no implicit assumptions about aspect relationships such as precedence. The compiler demands an explicit precedence specification when there is a potential overlap between the joinpoints that two aspects advise. Similarly, our system is type-safe from an aspectual point of view. The advice has to define the types of variables it obtains from the joinpoint (e.g. arguments). The compiler checks whether this type is indeed delivered by the joinpoint. This is in contrast to typical framework-based AOP approaches such as JBoss AOP [3], which do not provide similar type-safety.

4.3 Evolvability

Because of the improved separation of concerns, evolvability is also enhanced significantly. Every concern is confined in one module, making the addition/modification/removal of a concern easier. In addition, our system supports dynamic evolvability in order to encompass long-running workflows.

5. RELATED WORK

An aspect-oriented extension to BPEL4WS — AO4BPEL [12] — 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.

Currently, grid services are mostly composed manually, by writing programs in traditional programming languages (e.g. C and Java), which use libraries such as the ones provided by the Globus Toolkit [1] 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 this problem.

GridNexus [11] 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 services 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 [6] 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 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, dynamic workflow adaptation, and high-performance computing.

Therefore, we propose a new generation workflow system, which is specifically tailored for grid services. Although our system is targeted at grid services, some of its contributions can also be recuperated for web service workflow languages. For instance, we expect that the support for AOSD and dynamic workflow adaptation can be generalized to existing technologies for web service composition.

7. REFERENCES

[1] The Globus Toolkit. `http://www.globus.org/toolkit/`.

[2] GridCafé. `http://gridcafe.web.cern.ch/gridcafe/`.

[3] JBoss Aspect Oriented Programming. `http://www.jboss.org/products/aop`.

[4] The Open Grid Services Architecture (OGSA). `http://www.globus.org/ogsa/`.

[5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, editors. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, Heidelberg, Germany, 2004.

[6] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. 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, June 2004.

[7] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services version 1.1, May 2003. `http://www.ibm.com/developerworks/library/ws-bpel/`.

[8] J. Basney and M. Livny. Deploying a high throughput computing cluster. In R. Buyya, editor, *High Performance Cluster Computing: Architectures and Systems, Volume 1*. Prentice Hall, 1999.

[9] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran. WebFlow — a visual programming paradigm for web/Java based coarse grain distributed computing. *Concurrency — Practice and Experience*, 9(6):555–577, 1997.

[10] D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey, S. Weerawarana, and S. Winkler. Web Services Addressing (WS-Addressing). W3C Member Submission 10 August 2004, World Wide Web Consortium, August 2004. `http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/`.

[11] J. L. Brown, C. S. Ferner, T. C. Hudson, A. E. Stapleton, R. J. Vetter, T. Carland, A. Martin, J. Martin, A. Rawls, W. J. Shipman, and M. Wood. GridNexus: A grid services scientific workflow system. *International Journal of Computer & Information Science*, 6(2):72–82, June 2005.

[12] A. Charfi and M. Mezini. Aspect-oriented web service composition with AO4BPEL. In L.-J. Zhang, editor, *Proceedings of the 2nd European Conference on Web Services (ECOWS 2004)*, pages 168–182, Erfurt, Germany, September 2004. Springer-Verlag.

[13] B. De Win, W. Joosen, and F. Piessens. Developing secure applications through aspect-oriented programming. In R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 633–650. Addison-Wesley, Boston, 2005.

[14] M. D'Hondt and V. Jonckers. Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. In K. Lieberherr, editor, *Proc. 3rd Int' Conf. on Aspect-Oriented Software Development (AOSD-2004)*, pages 132–140. ACM Press, Mar. 2004.

[15] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, June 2002. `http://www.globus.org/alliance/publications/papers/ogsa.pdf`.

[16] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[17] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. 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, November 2001.

[18] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz. Programming the grid: Distributed software components, P2P and grid web services for scientific applications. *Cluster Computing*, 5(3):325–336, July 2002.

[19] N. Kavantzas, D. Burdett, and G. Ritzinger. Web Services Choreography Description Language version 1.0. W3C Working Draft 27 April 2004, World Wide Web Consortium, April 2004. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/`.

[20] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proc. ECOOP 2001, LNCS 2072*, pages 327–353, Berlin, June 2001. Springer-Verlag.

[21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. Technical Report SPL97-008 P9710042, Xerox PARC, Feb. 1997.

[22] F. Leymann. Web Services Flow Language (WSFL 1.0). IBM, May 2001.

[23] M. Lorch and D. Kafura. 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)*, pages 136–143, Berlin, Germany, May 2002.

[24] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(12):1053–1058, Dec. 1972.

[25] A. Popovici, T. Gross, and G. Alonso. Dynamic weaving for aspect-oriented programming. In G. Kiczales, editor, *Proc. 1st Int' Conf. on Aspect-Oriented Software Development (AOSD-2002)*, pages 141–147. ACM Press, Apr. 2002.

[26] M. Romberg. The UNICORE grid infrastructure. *Scientific Programming, Special Issue on Grid Computing*, 10(2):149–157, 2002.

[27] D. Suvée and W. Vanderperren. JAsCo: An aspect-oriented approach tailored for component based software development. In M. Akşit, editor, *Proc. 2nd Int' Conf. on Aspect-Oriented Software Development (AOSD-2003)*, pages 21–29. ACM Press, Mar. 2003.

[28] S. Thatte. XLANG — web services for business process design. Microsoft, June 2001. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`.

[29] W. M. P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, January/February 2003.

[30] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.

[31] W. Vanderperren, D. Suvée, and V. Jonckers. Combining AOSD and CBSD in PacoSuite through invasive composition adapters and JAsCo. In *Proceedings of Net.ObjectDays 2003*, pages 36–50, Erfurt, Germany, September 2003.