A Framework for Automatic Composition of Scientific Experiments: Achievements, Lessons Learned and Challenges

Luciano A. Digiampietri[1], José J. Pérez-Alcázar[1], Caio R. N. Santiago[1], Guilherme A. Oliveira[1], Adilson Khouri[1] and Jonatas C. Araujo[1]

[1]Escola de Artes, Ciências e Humanidades da Universidade de São Paulo - EACH-USP

digiampietri@usp.br

Abstract. *Scientific workflows management systems (SWMS) play a very important role in the e-Science. This paper presents the new functionalities of a SWMS that was originally developed fifteen years ago. These functionalities are focused on the automatic composition and execution of workflows, and the transparent use of local applications, Web Service and Java methods as the building blocks of the scientific experiments.*

1. Introduction

In the last years, several approaches were developed to facilitate the management of scientific workflows, including: the graphical creation and execution of workflows, the use of Semantic Web Services, the storage and track of workflows provenance data, the sharing of workflows over the Web and the (semi-)automatic construction of workflows (using, for example, artificial intelligence planning or recommendation techniques).

This paper aims to present achievements, lessons learned and current challenges in a framework for the management of scientific experiments. This framework is our extension of a project started in the nineties [Seffino et al. 1999], and, this paper presents the main functionalities developed in the last years, highlighting the automatic composition and execution of workflows, and the transparent use of local applications, Web Service and Java methods as the building blocks of the scientific experiments.

2. Basic Concepts

Workflows are now recognized as a crucial element of the e-Science infrastructure, working as the foundations where researchers can model, design, execute, debug, re-configure and re-run their experiments [Deelman et al. 2009].

The manual composition of experiments is an arduous and susceptible to errors task. Thus, automatic and semi-automatic solutions become popular in the academic and industry communities. There are different strategies for automatic composition of workflows, some use Artificial Intelligence (AI) techniques (such as planning or symbolic model checking), other use Markov Models, or even interface matching techniques.

An alternative way to classify the automatic composition is according with the kinds (or levels) of information that is available to the composition process [Digiampietri et al. 2011]. There are three main categories: (a) *syntactical* composition: when the only information available is the (syntactical) data types of the interfaces (matching inputs and outputs of the activities); it is typically used in approaches focused

on data transformation workflows; (b) *semantical* composition: when there are information about the data types and the semantic concepts associated with each input and output [Oh et al. 2007]. Domain ontologies are used to annotate the input and output data, allowing a more precise composition process; and (c) *functional* composition: it is the composition that considers the use of pre- and post-conditions (besides the information about inputs and outputs) [Shin et al. 2009], it is the most complete type of composition, where not only interface information is taking in account.

In this paper, the scientific experiments are defined as workflows composed manually or automatically (using AI planning). The composition technique considers all the available information (syntactic, semantic and functional) and the quality of the produced workflows strongly depends on the kind, amount and quality of the available information.

3. Historical Description of WOODSS Framework

The first version of WOODSS (*A Spatial Decision Support System based on Workflows*) [Seffino et al. 1999] was implemented on top of a commercial Geographic Information System to capture and store the interactions of the users, allowing sharing, updates and re-execution of experiments. WOODSS evolved to a generic workflow management system [Medeiros et al. 2005] to help scientists to specify, annotate, and share their models and experiments. The evolution of the framework included the storage of the workflows in databases (including input data and results), the semantic annotation and description of services as digital objects (easy to reuse and to execute) and a first approach in the automatic composition of services.

Some of the extensions of the framework include the storage of provenance data, the use of ontologies to describe data and services, and to allow the automatic composition of Semantic Web Services using Artificial Intelligence Planning, the development of traceability mechanisms and for helping in software component reuse [Digiampietri et al. 2007, Barga and Digiampietri 2008, Digiampietri et al. 2013, Zuñiga et al. 2014]. The next section presents the new features focused on the composition and execution of the scientific experiments, regarding real world experiments.

4. Current Framework

Figure 1 presents the framework architecture. It's general description was presented in [Digiampietri et al. 2007]. The previous version of the framework had four main limitations: only Web services were allowed as workflows activities; the automatic composition strategy did not used pre- and post-conditions; the execution of the workflows activities' was made sequentially using a third party workflow engine; the workflows could not be exported as executable code. This paper highlights the new functionalities and the implementation aspects of *Planner*, *Editor*, and *Executor* modules which overcome the limitations presented in the previous version of the system.

The current version of the framework uses three types of basic activities: Web Services, methods written in Java, and local applications. While the use of Web Service is very important to guarantee interoperability and to make easier the distributed execution, there are several problems and challenges when using only Web Services as the building blocks of workflows. One is the need for converting all the applications used by the researchers as Web Services. Obviously, it can be easily made by producing a container

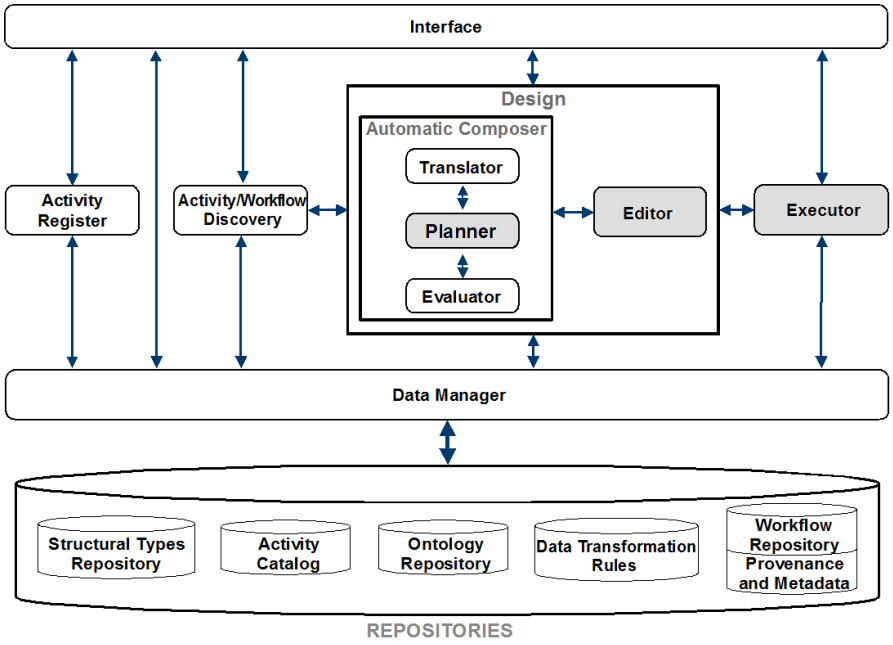

Figure 1. Architecture

Web Service which goal is to invoke a local application, but, even then, it is an arduous task when considering the dozens of applications daily used by the researchers.

In order to use third party Web Services in workflows system it is desirable that the services are described semantically. Unfortunately, different groups use different domain ontologies that are rarely compatible, thus, in order to use the semantic description of these services, it is necessary to match different ontologies (typically in a semi-automatic way) or to re-annotate the desired services. In this sense, Web Services are not as interoperable as one should demand. Web Services also require the transmission of input and output data throw the Web when the data and services are not in the same place. Whenever there are a great volume of data, or a low transmission rate, or even the date is confidential, the use or not of Web Services should be carefully considered.

In order to facilitate the use of the applications which the researchers already use, the proposed framework has a special kind of activity to run local applications. The user can easily register local applications as new workflows activities. In order to do this, the user needs to provide the name of the application, the path of the executable file, and the number and name of the parameters. Besides the regular inputs, two additional inputs are automatically inserted in this kind of activity: the names of two output files, one to store the standard output and other to store the error output (these two parameters are used as the outputs of this kind of activity). When registering any new activity the user is asked to annotate syntactically and semantically each activity's inputs and outputs.

The last type of activity supported by the framework is methods written in Java. There are *three* main goals of allowing this kind of activity. The first one was an empirical observed need for simple specific activities for data conversion, extraction or even any kind of simple processing required between some complex tasks. For example, after a genome assembly it was necessary to identify some metrics to be used as input of a genome annotation process. When using only Web Services, it was necessary to develop

a new Web Service to, for example, find a specific value inside one partial result file. Regular Java methods are easier to develop and more efficient than Web Services. This kind of activity is typically developed by a computer science specialist (undergraduate or graduate student, or a researcher) and can be registered in the system in the same way a Web Service or a local application. The unique difference is that, when registering a Java method, the user must indicate the *.class* file and the name of the method to be stored. The framework automatically extracts the interface information of the method using *Reflection*[1] and the user is asked to annotate semantically each method's inputs and output. If the method's interface is described using *Annotation*[2] then the framework will extract this information automatically.

The second advantage of allowing Java methods is the possibility of using shared memory in the execution of the workflows activities. Since the Executor Module was developed in Java, the activities which use Java methods can use as input and output references do objects in memory instead of copies of these objects.

The last main advantage of supporting Java methods as activities lays in the exportation of the workflow source code. One of the functionalities of the framework is to export the experiments as executable (or Java) code. Whenever all the activities of one workflow are Java methods there is the possibility of exporting the workflow as pure Java code (with no need of the exportation of the Workflow Engine module). The exported code will be simpler, facilitating edition or re-execution. Whenever the workflow to be exported contains Web Services, the resulting code imports the classes from the Workflow Engine in order to invoke the Web Services or call local applications. Even thus, the exported workflow can be executed outside the proposed framework (facilitating the sharing and reuse of scientific experiments).

The Compositor Modules uses SHOP2 [Nau et al. 2003] as planner. The use of this planner has two main advantages: (i) SHOP2 is a very efficient hierarchical task network planner able to work with large domains to produce the best or all the solution plans according with a cost function; and (ii) it is able to do the automatic composition using any kind of available data (syntactical, semantical or functional). The difference in the kind of data used lays on the domain and problem definition (so it is transparent to the planner). The framework produces the domain definition based on available data. Since all activities (Web services, local applications or Java methods) have at least the syntactical information about their interfaces, this information is always translated to the planner. Whenever there are semantic annotations of the interfaces, they are also used (allowing the composition of more specific and, probably, more useful plans). If no semantical data is provided, all inputs and outputs are annotated as belonging to the concept Thing, the most general concept in the domain ontologies. Functional information, when available, is translated as pre- and post-conditions to be used by the planner.

Even without any annotation of activities it is possible to take advantage of the automatic composition module. For example, in the current version of the framework, a user can use as input a list of Java classes and the system will convert each method to an activity and produce a SHOP2 domain definition using only the syntactical information obtained automatically from the classes. Thus, the user provides its input data and can

[1] http://java.sun.com/developer/technicalArticles/ALT/Reflection/

[2] http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html

request, as goal, the production of a set of data types or the execution of a given set of methods and the planner will try to produce a plan able to satisfy the request. A plan is an ordered set of tasks (activities) which execution achieves a given goal (solves the requested problem).

The Editor Module is responsible for the register of new activities and the graphical edition of workflows. Moreover, the Editor can receive a plan as input and convert it to a workflow. It is possible that the same plan can be converted into different workflows (since the plan is just an ordered set of activities). The implemented translation process, first tries to match the input of the last activities with the outputs of the previous ones, and then fill the unfilled activities' inputs with the input data passed by the user. The resulting workflow is presented to the user, so he/she can edit or ask for its execution.

The Executor Module is responsible for the execution of workflows. It supports all three kinds of activities, executing each workflow activity in a new thread considering data and control flows.

4.1. Basic Case Study

This section presents a simple case study in order to illustrate the translation from a plan to a workflow, the exportation of the workflow as executable code, the graphical version of the workflow and its execution.

As presented, a plan is an ordered set of tasks (activities) which execution achieves a given goal. Given a plan composed of four activities: *add*, *add*, *multiply*, and *print* (in this case, all activities are methods from a Java class). This plan can be imported with or without a list of input parameters. The code of the Plan Importation algorithm can be seen in the supplementary material[3]. Whenever the list of input parameters is not empty, the algorithm uses this information as input of the workflow activities. The list of input parameters is passed as a list of pairs: <data type, value>. In this case study, the list of input parameters has four elements: four data of type *double* and which values are, respectively, *2*, *3*, *7* and *9*.

The graphical view of the resulting workflow is presented in Figure 2. Each task/activity is represented by a rectangle. The inputs and outputs of each activity are represented by small rectangles in the left (inputs) and right (outputs) sides of each activity. The data flows are represented by black arrows linking an output to an input. In this workflow, the two *add* activities (the activities in the left side of the workflow) will be executed concurrently. The execution of this workflow produces the value *80.0*, corresponding to the result of the following operations: *(2+3)*(7+9)*. The workflow from Figure 2 can also be automatically exported as an executed code[4].

The example used in Figure 2 does not have any explicit control flow (only data flows). Figure 3 presents the workflow from Figure 2 with one additional control flow. Control flows are gray arrows which link one activity with other, meaning that the execution of one activity can start only after the execution of the other activity (regardless the absence of data flows). A more complex workflow, composed of three methods written in Java and two web Services can be seen in the supplementary material[5].

[3]http://www.each.usp.br/digiampietri/bresci2014/code1.png
[4]http://www.each.usp.br/digiampietri/bresci2014/code2.png
[5]http://www.each.usp.br/digiampietri/bresci2014/code3.png

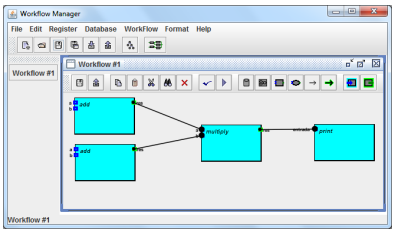

Figure 2. Study case workflow

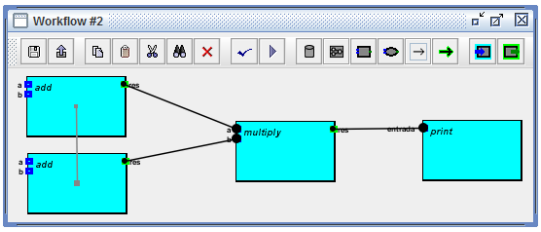

Figure 3. Workflow with a control flow

4.2. Achievements

Besides the toy examples used to validate or evaluate the framework, the current version of the framework is being successfully used in two real cases.

The first was managing scientific experiments to facilitate the use of non-technical users. The framework was used by biochemical users in order to analyze morphological characteristics of *Drosophila* embryos to evaluate the effects of the expressions of some genes in the formation of the anterior regions of the embryos [Andrioli et al. 2012].

In the second experiment, the framework was used as the basis for the development of a configurable and extensible environment for the management of sign language processing experiments [Digiampietri et al. 2012], and a set of specific workflow activities was developed including image segmentation and classification methods.

4.3. Lessons Learned

Web Services are a very useful and powerful technology, but using only this kind of activity can waste available applications and/or restrict the number of the framework users.

The automatic composition mechanisms can help users to obtain a composed workflow that achieves a given goal, but the majority of the scientists knows what tools they want to use, thus, semi-automatic construction of workflows may be more useful than the automatic one. For this reason, incorporate recommender approaches seems to be a logical evolution of the framework.

The planner and the recommender systems are as good as the workflows and activities database/catalog. Without a well-structured and well annotated set of activities and workflows none of these approaches will produce good results. Actually, it is very hard to find good workflows or semantic web services repositories. One of the biggest workflows open repository is *MyExperiment*[6] and it has only few hundred workflows (which are not semantically annotated).

4.4. Challenges

There are three main challenges that we started to face.

- The first is to build the framework as a set of loosely coupled module, so the modules can be easily replaced or be incorporated in other solutions. In order to do this, the interfaces of all modules are being formally specified.
- The second challenge regards usability. To facilitate the use of the framework, especially for non-technical users, it is necessary to improve the graphical interface and to develop more complex fault tolerance mechanisms.

[6] www.myexperiment.org/

- The third challenge is to combine automatic composition and recommender techniques, considering small datasets of workflows and annotated activities. More than only developing approaches which combine these techniques, it is very important to develop tools to facilitate activities and workflows annotation and sharing to incentive users to increase the repositories with high quality annotated activities/services and workflows.

5. Related Work

Some of the most used SWMS are *Kepler* [Altintas et al. 2004], *Triana* [Taylor et al. 2005] and *VisTrails* [Callahan et al. 2006]. A detailed review of the SWMS describing capabilities, limitation, and challenges can be seen in [Curcin and Ghanem 2008, Deelman et al. 2009, Lu and Zhang 2009].

The work presented in this paper differs from related work for allowing the easy combination, sharing and execution of basic activities that researchers already use in its local experiments. Moreover, it supports the combination of local applications, methods developed in Java and Web services, allowing the sharing of experiments as workflows or executable code (which does not require the installation or use of a SWMS). The planner used is an hierarchical planner which take advantage of syntactical, semantical, pre- and post-conditions information in order to automatically produce workflows.

6. Conclusions and Future Work

This paper presented the new functionalities of a scientific workflow management system which first version was developed 15 years ago. These new functionalities aims to facilitate the composition and execution process, combining the interoperability of Web Services with the traditional local applications and the methods developed in Java.

Moreover, the process of extracting information from activities was partially automated, so the user is asked only to complement the activities information (for example, annotating semantically the activities interfaces). The resulting plans produced by the planner are automatically converted to workflows which can be edited, executed or exported as executable code.

As future work, we intend to develop recommender techniques to help the user in the manual construction of workflows. Moreover, the graphical user interface will be improved to facilitate the management of workflows and the register of new activities.

Acknowledgments

The work presented in this paper was supported by FAPESP, CAPES and by CNPq.

References

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM'04*, pages 423–424.

Andrioli, L. P., Digiampietri, L. A., de Barros, L. P., and Machado-Lima, A. (2012). Huckebein is part of a combinatorial repression code in the anterior blastoderm. *Developmental Biology*, 361(1):177 – 185.

Barga, R. S. and Digiampietri, L. A. (2008). Automatic capture and efficient storage of escience experiment provenance. *CCPE*, 20(5):419–429.

Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., and Vo, H. T. (2006). Managing the evolution of dataflows with vistrails. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*, ICDEW '06, page 71.

Curcin, V. and Ghanem, M. (2008). Scientific workflow systems - can one size fit all? In *Biomedical Engineering Conference, 2008 - CIBEC2008*, pages 1–9.

Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.*, 25(5):528–540.

Digiampietri, L., Teodoro, B., Santiago, C., Oliveira, G., and Araújo, J. (2012). Um sistema de informação extensível para o reconhecimento automático de libras. In *SBSI 2012 - Trilhas Técnicas (Technical Tracks)*.

Digiampietri, L. A., ARAUJO, J. C., OSTROSKI, E. H., Santiago, C. R. N., and Perez-Alcazar, J. J. (2013). Combinando workflows e semântica para facilitar o reuso de software. *Revista de Informática Teórica e Aplicada: RITA*, 20:73–89.

Digiampietri, L. A., de Jesús Pérez Alcázar, J., and Medeiros, C. B. (2007). An ontology-based framework for bioinformatics workflows. *IJBRA*, 3(3):268–285.

Digiampietri, L. A., Pérez-Alcázar, J. J., e Freitas, R. S., Araújo, J. C., ÉricH. Ostroski, and Santiago, C. R. N. (2011). Uso de planejamento em inteligência artificial para o desenvolvimento automático de software. In *AutoSoft 2011*.

Lu, S. and Zhang, J. (2009). Collaborative scientific workflows. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 527–534.

Medeiros, C., Perez-Alcazar, J., Digiampietri, L., Pastorello, G., Santanche, A., Torres, R., Madeira, E., and Bacarin, E. (2005). WOODSS and the Web: Annotating and Reusing Scientific Workflows. *ACM SIGMOD Record*, 34(3):18–23.

Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, pages 379–404.

Oh, S.-C., Lee, D., and Kumara, S. R. T. (2007). Web service planner (wspr): An effective and scalable web service composition algorithm. *IJWSR*, 4(1):1–22.

Seffino, L., Medeiros, C., Rocha, J., and Yi, B. (1999). WOODSS - A Spatial Decision Support System based on Workflows. *Decision Support Systems*, 27(1-2):105–123.

Shin, D., Lee, K., and Suda, T. (2009). Automated generation of composite web services based on functional semantics. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):332–343.

Taylor, I., Shields, M., Wang, I., and Harrison, A. (2005). Visual grid workflow in triana. *Journal of Grid Computing*, 3(3-4):153–169.

Zuñiga, J. C., Pérez-Alcázar, J. J., Digiampietri, L. A., and Barbin, S. E. (2014). A loosely coupled architecture for automatic composition of web services applications. *International Journal of Metadata, Semantics and Ontologies*, 9(3):241–251.