Components and Service Farms

Gabriela Beatriz Arévalo, Zeina Azmeh, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, Sylvain Vauttier

Component and Service Farms

Gabriela Arévalo
LIFIA - Facultad de Informática (UNLP) - La Plata (Argentina)
garevalo@sol.info.unlp.edu.ar

Zeina Azmeh, Marianne Huchard, Chouki Tibermacine
LIRMM - CNRS UMR 5506 - Université de Montpellier II - Montpellier (France)
{*azmeh, huchard, tibermacin*}*@lirmm.fr*

Christelle Urtado, Sylvain Vauttier
LGI2P - Ecole des Mines d'Alès - Nîmes (France)
{*Christelle.Urtado, Sylvain.Vauttier*}*@mines-ales.fr*

I. Context

Software components and web services are software building blocks that are used in the composition of modern software applications. They both provide functionalities that require to be advertised by registries in order to be discovered and reused during software building processes[1] [1], [2], [3], [4], [5]. Building or evolving existing software implies assembling software components. This task is not trivial because it requires to select the adequate component or service that provides some part of the desired application functionality and connects easily (with minimum adaptations) to other selected components.

Within this context, we identified two main issues: (1) finding appropriate components from huge databases, and (2) creating and maintaining distributed applications.

Finding appropriate components from huge databases: A huge number of components already exist. For example, the Seekda[2] web service search engine has a catalog of more than 28.000 references, and the OW2 consortium[3] groups around 40 open-source projects in the domain of middleware technology. In Seekda, functionalities are rather directly exposed, but in projects, components are sometimes buried into application code repositories and should be properly extracted before being publicly advertised. However, in both cases the task of finding and adapting an appropriate component is hard and time-consuming, and this will surely worsen when more components become available.

Creating and maintaining distributed applications: There is a growing need for being able to create and maintain distributed applications assembled from third party components as network, middleware and deployment technologies now are mature enough. Reduced cost, increased quality and decreased dependence to a given provider also contribute to popularize this trend. Paradigms that are based on this principle are numerous: component-based development, web 2.0, mashups, cloud computing, software as a service, etc. State-of-the-practice technologies such as OSGi[4], that enables the deployment and redeployment of software (for example, in remotely administered internet boxes), or upcoming technologies such as Google Chrome OS[5], that aims to put web-apps at the center of net-books' operating system, also illustrate this trend.

II. Innovative registries

In this context, it is necessary to design innovative software component registries to advertise components and assist users when they need to search, select, adapt and connect a component to others [6], [7]. We even could think of the automation of these tasks as much as possible, in order to adapt to open and dynamic contexts (remotely administered embedded devices, pervasive computing, open and extensible applications, mobile computing, etc.).

Then, the challenge consists in proposing an online architecture for a component service registry, with two functionalities (a) gives efficient access to adequate components, and (b) provides life-cycle long support to developers and applications (in automatic mode). This registry will be a platform for developers to share their knowledge and experiences on the components they use (what runs and what does not, what are ideal contexts for running some given component, what adaptations have already been performed or tested on some component, what are user ratings on components, etc.), and improve development efficiency as well as running software reliability or liveliness.

As an extension of the component search engine, we think of a *component farm*, where components could be either physically located (component repository model) or solely referenced in an adequately organized component

[1] In the following, we will often use the term *components* as a generic name for both web services and software components.

[2] http://webservices.seekda.com/

[3] http://www.ow2.org/

[4] http://www.osgi.org

[5] http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html

advertisement directory). The component farm (*cf.* Fig. 1) should offer multiple views on components and tailored efficient retrieval mechanisms.

Developed software applications that use components from the farm could either simply use the farm's public facilities or register to benefit from increased community-related capabilities. In the latter case, the added value would be provided by the exchange of component-related data between registered software applications and the farm, in both directions:

- *from software applications to the farm.* Applications could register components developed for their own purposes into the farm directory or share usage information on components they have previously retrieved from the farm, etc. This means that developers should contribute as in collaborative web.
- *from the farm to software applications.* The farm should provide several operations supported by efficient tools for searching and selecting components, providing usage feedback on components, informing the developer of other developers' experiences regarding new products, found problems with components, problem solving issues, possible adaptations, etc.

Our proposal focuses on two challenging issues, each of which is developped in one of the followig sections: (1) the adequacy and efficiency of the organization of the component directory, (2) the proposal of tailored support to applications.

III. Efficient organization

Many components exist but are not always available in public servers/directories. Providing abstract views and adequate services, a global organization could help to stimulate their sharing and be profitable increasing cross-fertilization between projects. The first aim of the registry is to efficiently organize components for several development and maintenance tasks. The components can be physically contained in the repository or just accessible from the directory, through hypertext links, from other external locations. An efficient organization is based on an efficient classification of the components. For example, some approaches [8], [9], [10], [11], [12], [13] have already investigated formal concept analysis (FCA) [14], [15], an approach that rigorously classifies data in structures that have strong mathematical properties. Other approaches should nevertheless be studied.

Several aspects can be used to classify components: syntactical, semantic and pragmatic [16], [17]. All of them could be used complementarily. At the syntactical level, component external descriptions include ports, interfaces, functionality signatures and parameter types. At the semantic level, components can be documented (with plain or structured text, with interaction protocols, etc.), described by keywords (either normalized through ontologies, for example, or not), and by any information that conveys the meaning of the component (for what purpose it has been developed, by whom, etc.). At the pragmatic level, components are documented by information that provides feedback from its usage, in assemblies, choreographies or orchestrations: which functionalities are used, which functionalities are never used, how components are connected (with which adaptations), good and bad experiences, etc.

IV. Application support

The second aim of the registry is software application support. Applications using the components from the registry are invited to register so as the directory can receive feedback. Thus, the directory stores information on which application uses which components and documentation about this use, in order to share usage information among developers.

The registry memorizes the manual or automatic adaptations that are necessary when a faulty component is replaced by another. This information is used to optimize future replacements and is capitalized to be used by other registered applications.

The registry prepares backups (sets of possible substitutes) for each used component in order to ensure rapid repairing of a software application in case one of its components fails [18]. These backups are organized in small classifications for efficiency purposes. They can be used manually or sometimes automatically in restricted cases where either exact matching between the used component and its potential substitute exists, or automatic adaptations are known and can be applied. These backups can be stored inside the repository or uploaded on demand by applications. Backups can also be used by designers to make the software evolve when the application designer wants to add extra functionalities or improve the software quality attributes.

Candidate backup components and possible adaptations are dynamically updated, as components become unavailable or are upgraded. Software applications that use them are also dynamically informed of these updates.

V. Starting point

In previous work, we have started to imagine partial solutions to the issues identified here.

- We have developed an automated process for classifying components from their external descriptions. This process is based on type-theory (we only use syntactic information) and uses FCA to iteratively build lattices that provide functionality signature classifications, interface classifications and component classifications [8], [9], [10].
- We have prototyped the CoCoLa tool [10] that implements the aforementioned process. Thanks to a pivot meta-model, component descriptions from various formats are translated into comparable models (instances of the common meta-model). These descriptions are then processed to build context tables and lattices.

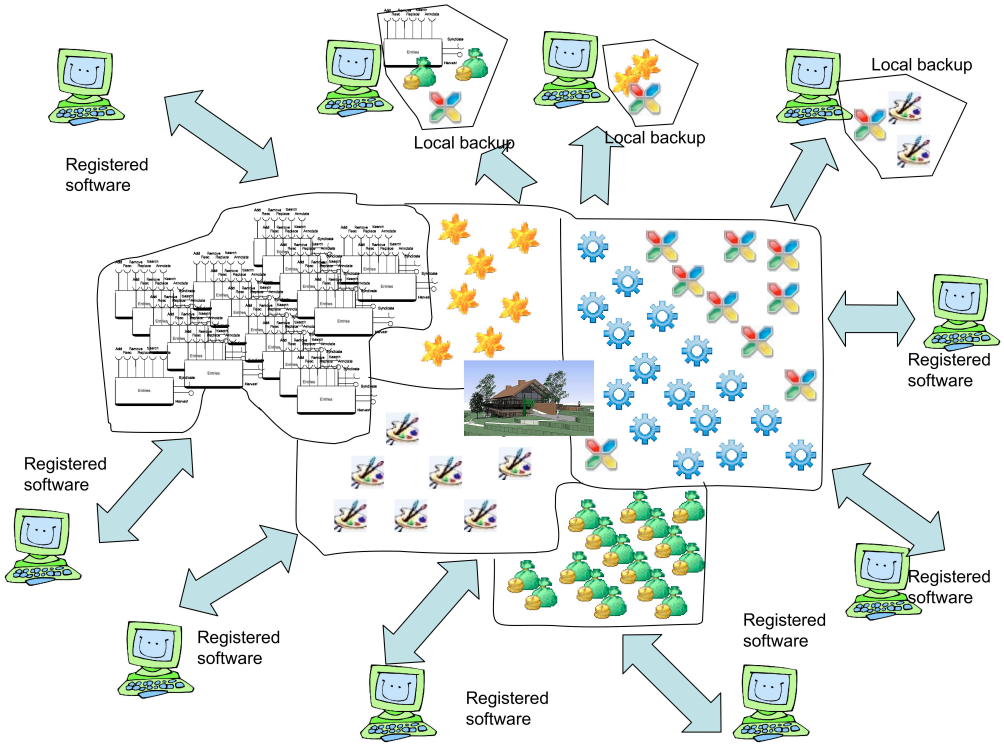

Figure 1. Overview of the component farm

Experiments have been run on the Dream[6] component library (that comes from a real-world component-based framework). They show the feasibility of our approach as the produced lattices enables us to identify possible component substitutions and provides a readable component classification.

- We also have proposed an approach based on formal concept analysis (FCA) for classifying web services [19], [18]. A web service lattice reveals the invisible relations between web services in a certain domain, showing the services that are able to replace other ones. This facilitates service browsing, selection and identification of possible substitutions. We explained how to exploit the resulting lattices to build web services orchestrations and support them with backup services.

VI. Main challenges

There still are numerous challenges to overcome in order to build such a component farm:

- combine syntactical, semantic and pragmatic views of components in order to be able to efficiently search and select appropriated components (pertinence of the classification).
- automate component indexing and classification as well as feedback collection in order not to put the burden on component developers or component users. This task could need to exploit automatically all the available semantic information available (such as functionality names, documentations, interaction protocols, etc.) using text mining techniques [20].
- try and capitalize coarser grained software parts by classifying whole component assemblies,
- think about building techniques and browsing techniques for each of the provided classifications but also inter-classifications.
- propose replacement scenarios and tools,
- define the services that would motivate users in collaboratively providing usage feedback information,
- find means to conduct early experimentations of the component farm. This is not easy as very few component directories are available online, as for now.

As a response to the globalization challenge, software engineering has the capability to positively react in providing software component catalogs, thus increasing software quality while decreasing costs and time-to-market. As an immaterial product market-place, the software component industry benefits from great assets: software components must not be stocked, they can be transported at no cost and instantaneously, they are not perishable. To take advantage of this, efforts should be focused not only on development,

[6]http://dream.ow2.org/

as traditionally done, but also on better diffusion, sharing, deployment and maintenance. We believe that component farms could contribute to this objective.

REFERENCES

[1] W. Hoschek, "The web service discovery architecture," in *Int'l. IEEE/ACM Supercomputing Conference (SC 2002)*, I. C. S. Press, Ed., 2002.

[2] OMG, "Trading Object Service Specification (TOSS) v1.0," 2000, http://www.omg.org/cgi-bin/doc?formal/2000-06-27.

[3] *ebXML Registry Services Specification (RS) v3.0*, http://www.oasis-open.org/, May 2005.

[4] L. Clement, A. Hately, C. von Riegen, and T. Rogers, "Uddi version 3.0.2. uddi spec technical committee draft , dated 20041019. http://uddi.org/pubs/uddi_v3.htm," Tech. Rep. [Online]. Available: http://uddi.org/pubs/uddiv3.htm

[5] Information Technology Open Distributed Processing, "ODP Trading Function Specification ISO/IEC 13235-1:1998(E)," December 1998, http://webstore.iec.ch/preview/info_isoiec13235-1%7Bed1.0%7Den.pdf.

[6] L. Iribarne, J. M. Troya, and A. Vallecillo, "A trading service for COTS components," *The Computer Journal*, vol. 47, no. 3, pp. 342–357, 2004.

[7] S. Dustdar and M. Treiber, "A view based analysis on web service registries," *Distributed and Parallel Databases*, vol. 18, pp. 147–171, 2005.

[8] G. Arévalo, N. Desnos, M. Huchard, C. Urtado, and S. Vauttier, "Precalculating component interface compatibility using FCA," in *Proceedings of the 5^{th} international conference on Concept Lattices and their Applications (CLA 2007), CEUR Workshop Proceedings Vol. 331*, J. Diatta, P. Eklund, and M. Liquière, Eds., Montpellier, France, October 2007, pp. 241–252. [Online]. Available: http://ceur-ws.org/Vol-331/

[9] ——, "FCA-based service classification to dynamically build efficient software component directories," *Int. Journ. of General Systems*, vol. 38, no. 4, pp. 427–453, May 2009.

[10] N. A. Aboud, G. Arévalo, J.-R. Falleri, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "Automated architectural component classification using concept lattices," in *In proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture 2009 (WICSA/ECSA'09)*. Cambridge, UK: IEEE Computer Society Press, September 2009.

[11] A. M. Zaremski and J. M. Wing, "Specification matching of software components," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, no. 4, pp. 333–369, 1997.

[12] B. Fischer, "Specification-based browsing of software component libraries," in *Proc. of the 13^{th} IEEE int. conf. on Automated Software Engineering (ASE'98)*, 1998, pp. 74–83.

[13] B. Sigonneau and O. Ridoux, "Indexation multiple et automatisée de composants logiciels orientés objet," in *AFADL — Approches Formelles dans l'Assistance au Développement de Logiciels*, J. Julliand, Ed. Besançon, France: RTSI, Lavoisier, Juin 2004.

[14] M. Barbut and B. Monjardet, *Ordre et Classification*. Hachette, 1970.

[15] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*. Springer, 1999.

[16] M. Á. Corella and P. Castells, "Semi-automatic semantic-based web service classification," in *Business Process Management Workshops*, ser. LNCS 4103, J. Eder and S. Dustdar, Eds. Springer, 2006, pp. 459–470.

[17] M. Bruno, G. Canfora, M. D. Penta, and R. Scognamiglio, "An approach to support web service classification and annotation," in *Proc. of the IEEE Int. Conf. on e-Technology, e-Commerce and e-Service (EEE'05)*, 2005, pp. 138–143.

[18] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "Using concept lattices to support web service compositions with backup services," in *To appear in Proceedings of the 5^{th} International Conference on Internet and Web Applications and Services (ICIW 2010)*, Barcelona, Spain, May 2010.

[19] ——, "WSPAB: A tool for automatic classification & selection of web services using formal concept analysis," in *Proceedings of the 6^{th} IEEE European Conference on Web Services (ECOWS 2008)*. Dublin, Ireland: IEEE, November 2008, pp. 31–40.

[20] J.-R. Falleri, Z. Azmeh, M. Huchard, and C. Tibermacine, "Automatic tag identification in web service descriptions," in *To appear in proceedings of the International Conference on Web Information Systems and Technology (WEBIST'10)*, Valencia, Spain, April 2010.