UNIVERSAL EXTRA-FUNCTIONAL PROPERTIES REPOSITORY [*]
Model Overview and Implementation

Kamil Ježek
Department of Computer Science and Engineering, University of West Bohemia Pilsen
Pilsen, Czech Republic

Keywords: Extra-functional, Repository, Component, Services, Meta-model.

Abstract: The current Internet is still more used for accessing data through remotely invoked services. Although it allows rapid development of new systems, it still has some limitations. This paper addresses extra-functional properties that must be taken into account to develop systems concerning calls of remote services. Since the services provide general functionality among worldwide applications, their extra-functional properties must be also generally accessible. Hence this paper targets less explored area of exchanging extra-functional properties via the Internet. For that reason we propose a model as well as an implementation of a universal repository that stores and accesses extra-functional properties. Main contribution of such repository is that the services use common extra-functional properties from one repository and thus different vendors work with the same and compatible properties.

1 INTRODUCTION

Nowadays software systems still increase in their size which is still more difficult to manage by traditional means. Innovating techniques such as component based programming or Service Oriented Architectures (SOA) tend to prevent creation of monolithic systems. They typically compose final systems from verified parts. A fundamental of component programming is to develop a system from pre-existing components while SOA tends to let applications invoke services available via the Internet. An idea of both techniques is, however, to use pieces of third-party functionalities.

Although SOA and components are reaching theirs maturity, there are still issues that prevent users to fully benefit theirs advantages. First of all, a user must verify whether services and components are compatible with the rest of the system. A considerable improvement of the verification process is taking extra-functional properties into account. As long as the verification works with extra-functional properties, the verification more closely reaches user expectations.

A lot of approaches have been developed to deal with extra-functional properties, but some issues have not been addressed yet. The worldwide sharing of the services must also consider a worldwide sharing of extra-functional properties. There are a lot of issues concerning semantics of extra-functional properties and theirs values. Obviously, a functionality must remain correct among organizations. On the other hand, extra-functional properties differ in different domains and contexts – e.g. some values are important in one domain, but vaguely important in another one. In addition, once vendors enhance their systems by extra-functional properties, the properties must be understandable by all participating vendors.

1.1 Goal of the Paper

This paper first introduces our approach that addresses several issues concerning worldwide exchange of extra-functional properties. Namely we propose (1) a model that allows to share extra-functional properties among organizations and (2) a model that deals with context dependency of extra-functional properties. We second introduce a tool we have implemented as a prove of the concept. The tool aims at improving exchange of extra-functional properties among worldwide vendors.

The rest of this paper is organized as follows: Sec-

[*]This work was supported by the Grant Agency of the Czech Republic under grant number 201/08/0266 “Methods and models for consistency verification of advanced component-based applications”.

tion 2 first overviews our previous work pressented in rather formal terms and then overviews other approaches concerning extra-functional properties. Section 3 introduces our approach presented as the metamodel. In Section 3.4 we state some worth reading details about the implementation.

2 RELATED WORK

This work has been based on more theoretical fundamentals we proposed in (Jezek et al., 2010). In that paper we presented formalization of extra-functional properties and system of registries that addresses context-dependency of these properties. Although our previous research was only for software components, the approach is not restricted to components and is ideally suitable also for SOA. A similarity of components and SOA is obvious: components communicate through services that may run in different computational environment.

A lot of other works create specialized languages for writing extra-functional properties as an ordinal source code. Let us name NoFun (Franch, 1998) that distinguishes between simple and derived extra-functional properties, CQML (Aagedal, 2001) that serves as a complete extra-functional language, CQML+ (Röttger and Zschaler, 2003) that explicitly takes a runtime environment dependency into account. Other rather specialized languages are: HQML (Nahrstedt and Wichadakul, 2001), TADL (Mohammad and Alagar, 2008), or SLAng (Lamanna et al., 2003) specialized especially for service level agreement. An advantage of such approaches is that they develop an idea of what should an extra-functional property stands for. On the other hand they barely target context dependency of values and exchange of extra-functional properties. Developing our approach, we consolidate typical features of extra-functional properties into one general and unified repository of extra-functional properties.

In a field of components, there are some component models that concern extra-functional properties as a part of their models. Let us name: Palladio (Becker et al., 2009) that targets mainly performance characteristics, Robocop (Muskens et al., 2005; Bondarev et al., 2006) for real-time characteristics, or ProCom (Sentilles et al., 2009). An interesting idea in ProCom are registries that stores EFPs. The main reason for introducing registries is to gather attribute types.

3 EXTRA-FUNCTIONAL REPOSITORY TOOL

To achieve a comparability of extra-functional properties (EFPs), they must be unified. For that reason we propose a repository that accesses common EFPs via the Internet. Hence misunderstanding of EFPs meaning is prevented as long as all participating organizations use EFPs from this unified repository. In addition, any third-party components and services may be compared because they use the properties with the same semantics.

3.1 Approach Overview

Figure 1 overviews a general mechanism of the approach. All software parts communicate via the Internet. At this point, we have added a generally accessible repository of EFPs. All systems and their subparts must use only EFPs from the repository. It is a noticeable improvement that prevents any usage of incompatible properties. Once the properties are fixed in the repository they have the same meaning among worldwide vendors.

Each component or service assigns an EFP via its name first, then concrete definition is looked up from the repository when it is needed. It forces vendors to use the same semantics for properties with the same names.

3.2 User Roles

Figure 1 furthermore shows three roles of typical users.

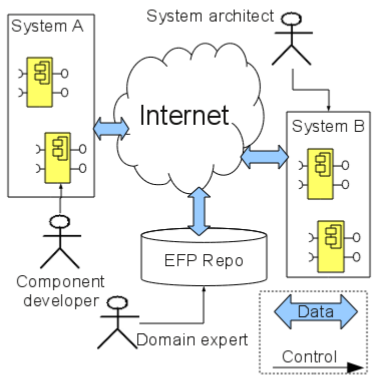

Figure 1: Overview.

There is a domain expert who role is to fill in the repository. He defines existence of EFPs and states their names, domain types and their comparing methods. Exact structure of EFPs we have already mentioned in (Jezek et al., 2010). We assume a perdomain repository, because there would be probably impossible to consolidate EFPs through all domains. We assume a set of domains (e.g. domain of schools, automotive, libraries) instead.

Another role is a component developer who prepares each component or service. He takes EFPs from the repository and attaches them to services. Since all component developers use properties from the same repository, they attach compatible properties.

Finally, the role of a system architect represents a user who composes a final system (composes components or services). He uses EFPs to estimate behavior of the final system. His expectations of the final system are expressed in terms of the properties from the repository.

3.3 Meta-model of EFPs Repository

Once extra-functional properties are used in worldwide applications a heterogeneity of environments must be taken into account. Addressing this heterogeneity we have developed a layered repository of EFPs. We discus it in formal terms in (Jezek et al., 2010). Figure 2 shows an implemented meta-model of the repository that we call Registry.

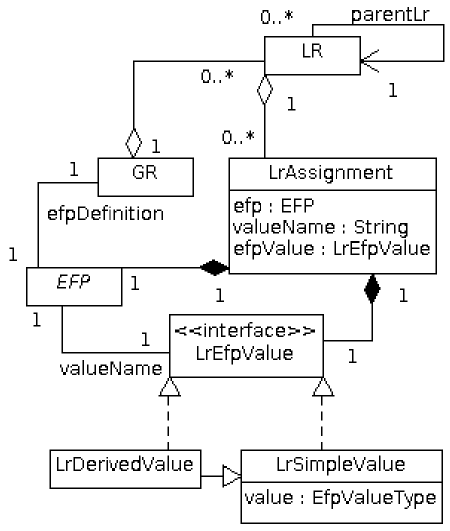

Figure 2: Registry.

The class `GR` represents the top level storage of EFPs – Global registry (Jezek et al., 2010). It stores instances of EFPs. For that reason it contains EFPs that are valid for a whole domain and distinguished by names. Global registry does not contain concrete values because they typically differ in different environments.

The class `LR` is called Local registry (Jezek et al., 2010) and encapsulates context dependent values addressing the heterogeneity of environments. Each computational environment (e.g. mobile phones, laptops, desktops, servers) must define its Local registry with concrete values.

The class `LrAssignment` assigns values to EFPs via names in Local registry. An assignment is a tuple concerning an EFP, a name and a value (Ježek, 2010), which assigns the value to the property by the textural name.

Considerable benefit of the assignment of values via textural names is that continuous intervals may be divided into disjunctive sets of named intervals. Hence values are considered as the same as long as they belong to the same interval (e.g. let us assume a property “response_time“ with the named interval $\{0;500\}ms$ in Local registry of small portable devices. Then all concrete values as $100ms$, $200ms$ or $300ms$ are considered as the same because they all belong to one group).

The other advantage is that this solution encapsulates context-dependent values denoted by names. The names do not change among environments while concrete values differ. E.g. a value labeled as a “high“ concerns very different numbers for laptop computers and mobile phones. However a developer think in more abstract terms (a meaning of “high” is obvious) rather than depending on concrete numbers.

Finally, two classes `LrSimpleValue` and `LrDerivedValue` denote assigned values for a simple or a derived property respectively. They are briefly explained in (Jezek et al., 2010) although their distinction is not so important for this paper.

3.4 Implementation

As a prove of the concept we have implemented the repository presented above.

To achieve an overall goal of the work we used a client-server architecture to allow the accessibility of the repository via the Internet. The server side application is a MySQL database server that has been selected as a storage of data. We have selected a database system because it allows to modify access rights, contains synchronization means, locks and a lot of other features that are useful for the concurrently accessed repository.

The client side application is a Java fat client that uses JFC Swing to create a graphical user interface. The client uses JDBC driver to access the remote database server.

A base skeleton of the application has been created from the meta-model in Figure 2 and the business logic has been added manually.

Let us note that the application is still a prototype that is capable of editing EFPs by a domain expert via the graphical user interface, however we are still working on a public interface that would provide EFPs to other applications via remote sockets.

4 CONCLUSIONS

In this paper we have dealt with components and Service Oriented Architecture. We have specifically targeted exchange of extra-functional properties among organisations.

We have proposed the universal repository of extra-functional properties that is accessible via the Internet. We have introduced an idea of how should such repository work first, then we have introduced the meta-model based on our previous work. Finally, we have highlighted some aspects of the implementation we did as the prototype application.

Although we have developed the prototype application, we still have open issues. In the future we would like to add a public interface that accesses extra-functional properties to remote systems.

Furthermore, there are some research issues that have not been addressed yet. The system of Local registries assumes that each feature assigns values for all contexts. Unfortunately, in reality a lot of registries may exist and it would be tedious work to prepare values for each service and Local registry. A solution may be functions ($v_y = f(v_x); v_y \in LR_1, v_x \in LR_2, LR$ is Local registry) that re-scale values automatically.

ACKNOWLEDGEMENTS

At this point of the paper, we would like to thank to students J. Chlad, L. Vlček, T. Sojka, J. Ryšavý, L. Kaloušková M. Bečvář, M. Štulc, J. Šváb, T. Pokrývka for the implementation of the presented prototype.

REFERENCES

Aagedal, J. Ø. (2001). *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo.

Becker, S., Koziolek, H., and Reussner, R. (2009). The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3 – 22. Special Issue: Software Performance - Modeling and Analysis.

Bondarev, E., Chaudron, M. R., and de With, P. H. (2006). Compositional performance analysis of component-based systems on heterogeneous multiprocessor platforms. In *Proceedings of Euromicro conference on Software Engineering and Advanced Applications*, pages 81–91. IEEE Computer Society.

Franch, X. (1998). Systematic formulation of non-functional characteristics of software. In *Proceedings of International Conference on Requirements Engineering (ICRE)*, pages 174–181.

Ježek, K. (2010). A complex meta-model for extra-functional properties concerning common data types their comparing and binding. In *Second World Congress on Software Engineering (WCSE 2010)*. (submitted to review).

Jezek, K., Brada, P., and Stepan, P. (2010). Towards context independent extra-functional properties descriptor for components. In *Proceedings of the 7th International Workshop on Formal Engineering approches to Software Components and Architectures (FESCA 2010)*.

Lamanna, D. D., Skene, J., and Emmerich, W. (2003). Slang: A language for defining service level agreements. *Future Trends of Distributed Computing Systems, IEEE International Workshop*, 0:100.

Mohammad, M. and Alagar, V. S. (2008). TADL - an architecture description language for trustworthy component-based systems. In *ECSA '08: Proceedings of the 2nd European conference on Software Architecture*, pages 290–297. Springer.

Muskens, J., Chaudron, M. R., and Lukkien, J. J. (2005). *Component-Based Software Development for Embedded Systems*, chapter A Component Framework for Consumer Electronics Middleware, pages 164–184. Springer Verlag.

Nahrstedt, G. and Wichadakul, Y. (2001). An xmlbased quality of service enabling language for the web.

Röttger, S. and Zschaler, S. (2003). CQML+: Enhancements to CQML. In Bruel, J.-M., editor, *Proc. 1st Int'l Workshop on Quality of Service in Component-Based Software Engineering, Toulouse, France*, pages 43–56. Cépaduès-Éditions.

Sentilles, S., Stepan, P., Carlson, J., and Crnkovic, I. (2009). Integration of extra-functional properties in component models. *12th International Symposium on Component Based Software Engineering (CBSE 2009), LNCS 5582*.