CREATIVE MATH. & INF. **Online version at** http://creative-mathematics.ubm.ro/
17 (2008), No. 3, 481 - 486 **Print Edition: ISSN 1584 - 286X Online Edition: ISSN 1843 - 441X**

Dedicated to Professor Iulian Coroian on the occasion of his 70^{th} *anniversary*

Component classification criteria for a platform - independent component repository

SIMONA MOTOGNA, IOAN LAZĂR, BAZIL PÂRV, ISTVAN CZIBULA, AND L. LAZĂR

ABSTRACT. Component repository is an essential part of a framework for software component definition, validation, and composition that we have developed so far. The components are stored into the repository such that search and retrieve operations will be as efficient as possible. We propose some classification criteria for components based on their signatures and functionalities. These criteria also take into consideration the specifications of the platform-independent component model for dynamic execution environment.

1. BACKGROUND

In component based software development reuse is a driven principle. The whole idea of developing component based applications is to integrate components from different parts in the architecture, with integration, adaption and modification rules. Component's degree of reuse depends on its generality, while the easiness in identification, understanding, and use is affected by the component specification.

In our opinion, the main component based development's challenge is to provide a general, flexible and extensible model, for both components and software systems. This model should be language-independent, as well as programming-paradigm independent, allowing the reuse at design level.

Our solution, ComDeValCo [7] - a conceptual framework for Software Component Definition, Validation, and Composition had started by proposing a way of defining executable models that could be turn in software components. After such components are defined, they are ready to be used at a design level. Such definitions should be stored in some way in order to be ready for reuse. The component repository will represent the persistent part of the framework, containing the models of all fully validated components.

Constituents of the conceptual framework are: the modeling language, the component repository and the toolset. Any model of a software component is described by means of a modeling language, programming language-independent, in which all modeling elements are objects. The component repository stores and retrieves valid component models. The toolset is aimed to help developers to define, check, and validate software components and systems, as well as to provide maintenance operations for the component repository.

Received: 20.09.2008. In revised form: 8.01.2009. Accepted: 26.05.2009.
2000 *Mathematics Subject Classification.* 68N30, 68P99.
Key words and phrases. *Repository, executable models.*

The rest of the paper is organized as follows: the next section discusses the state-of-the-art in repository architecture, Section 3 presents the description and the specification of the executable models to be stored, together with the repository metamodel. In the end, we highlight some conclusions and future work.

2. Related work

The complexity and the multidisciplinarity of the current software applications force the developers to search and integrate different components, from different parts in the application. But this distributed design generates difficulties associated with information sharing and integration. While share product knowledge has increase, it is still inadequate for developing complex products by distributed design teams [4].

As executable UML models have become more and more popular in software development based on extensive modeling, the components' repositories must be adapted to support this type of software development.

There is an extensive research and commercial interest in the domain of component repositories, but still they are not as performant as we would expect, due, in principal to factors like [8]: *extraction of information* that the repository should contain is mostly manually. Regardless of the used mechanism the characteristics of the components must be specified by the author; *search/retrieve techniques*: the repository user is not always satisfied with the result of the search. Usually several searching or query refinement are needed and still the user has to decide from several results which will be the best suited component that he / she needs.

As a result, it is possible that interrelated components may exist and be useful, but they have not been found by the query. The main reason for this situation is that the repository intern description does not contain semantic information about the components and semantic relations between components.

One particular framework which supports dynamic availability, and reconfiguration of components is the OSGi framework [6] which offers a service-oriented component model. OSGi components are bound using a service-oriented interaction pattern, and their structure is described declaratively. The OSGi Service Platform specification provides an open, common architecture for different types of users, such as service providers, developers, software vendors, enabling the development, deployment and management of services in a coordinated way. OSGi targets Java framework, in which the deployed applications are called bundles.

A framework which offers a simple solution for publishing and retrieving information is the ebXML Registry Information Model [5], [1]. This information system securely manages any content type and the standardized metadata that describes it. The ebXML Registry provides a set of services that enable sharing of content and metadata between organizational entities in a federated environment. The registry can be customized and used for specific domains and applications, by mapping a domain specific information model to the ebXML Registry Information Model.

These are basically the most two used approaches when designing a repository. The two infrastructures have a lot of similarities, and the differences are not very significant: usually they differ in the way the information regarding the repository objects is represented.

There are several tools that use one of these models for the infrastructure of a problem-specific repository.

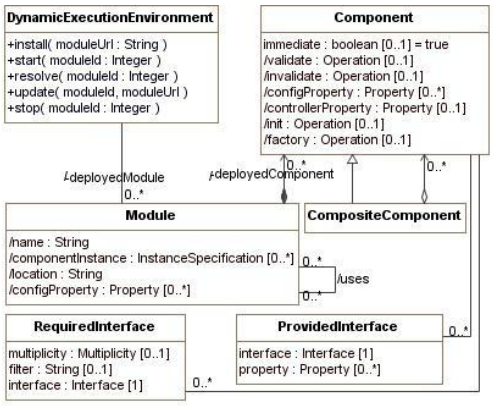

Figure 1. *i* COMPONENT metamodel

3. SPECIFICATION BASED CLASSIFICATION OF COMPONENTS

3.1. **iCOMPONENT description and specification.** The platform independent component model for dynamic execution environments, called iCOMPONENT [3] aims to simplify the component development by allowing developers to concentrate only on implementing the business logic of the component and then to configure declaratively the component deployment. It is intended to be a framework which supports dynamic availability, and reconfiguration of components, in the style of OSGi [6] and iPOJO [2] principles.

The UML metamodel of iCOMPONENT, as depicted in Figure **??**, highlights its constituents:

• *Dynamic Execution Environment* - represents an execution environment that provides capabilities for dynamic availability, reconfiguration, and composition of components;

• *Module* - which extends the UML Artifact metaclass and represents the unit of deployment. The set of model elements that are manifested in the artifact is indicated by the manifestation property of the Artifact;

• *Component* - extends the UML metaclass Class (from StructuredClasses) and represents a component type. By extending the metaclass Class, *Component* may have methods and attributes, and also may participate in associations and generalizations.

• *ProvidedInterface* - associated with each *Component* may have an associated set of properties, each property having a name, a type (optional), and a value. The set of properties can be used in the process of binding components;

• *RequiredInterface* - may indicate a filter property which specifies a query and will be used to query for services that satisfy the required properties.

3.2. **Repository metamodel.** Several repositories function in such a way that they return the list of all objects from the repository, and it is the user task to determine which he/she needs or, in other words, no searching is implemented in the repository structure. The searching operation would be more efficient if it will take into account a classification, such that when looking for a particular component only a part of the repository will be searched.

In conclusion, when designing the repository architecture, we consider that at the moment a component is stored, we will associate a classification scheme to it, and correspondingly a classification hierarchy can be built. The classification method is directly related to the search and retrieve mechanism that the repository will follow.

The repository will store iCOMPONENTs, represented as *DComponent*s. The repository infrastructure for platform-independent executable components follows the ebXML Registry Information Model [5] and is adapted to ComDeValCo structure. Figure 2 gives a simple description of classes and their relationships.

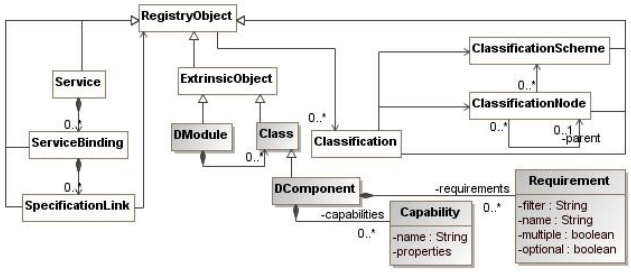

Figure 2. Repositoty infrastructure

According to OASIS/ebXML RIM Specification [5], *ExtrinsicObject*s provide metadata that describes submitted content whose type is not intrinsically known to the Registry and therefore MUST be described by means of additional attributes. Since the registry can contain arbitrary content without intrinsic knowledge about that content, *ExtrinsicObject*s require special metadata attributes to provide some knowledge about the object (e.g., mime type).

DModule is the unit of the repository structure and will be uniquely identified. Any *DModule* is classified according to any number of *Classification* instances. A *Classification* instance references a *ClassificationNode* instance, which represents a value within the *ClassificationScheme*. Consequently, the *ClassificationNode* instances may create a classification hierarchy in the following way: a node may have zero or at most one parent, and may have zero or more descendants.

Based on the iCOMPONENT metamodel for each *DComponents* stored in the repository, its *Capability* can provide information about the domain in which the component may be used, and consequently help the classification process. Any *DComponent*, such as classes, components or interfaces will be enhanced with classifiers when stored into repository. We may associate several classification criteria to a *DComponent*, depending on its functionality. Thus, when searching

into the repository such a component may be returned in two different results, based on different searching criteria.

In order to illustrate how such a classification works, we consider a simple case study, of a system that stores information about different products (code and description) and there exists a pricing strategy interface, that will be used for implementing discount computations, as shown in Figure 3.

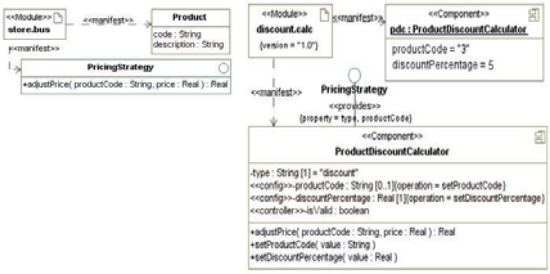

Figure 3. Components storing information

As a proof of concepts, we adapt and extend the classification scheme from ebRIM, including some new categories, based on the application domain from our example, as shown in Figure 4. Any new category that is added into the classification hierarchy must be a descendant of the *ExtrinsicObject*.

The classification scheme and the considered example highlight the fact that more than one classifier may be associated to a component from the repository.

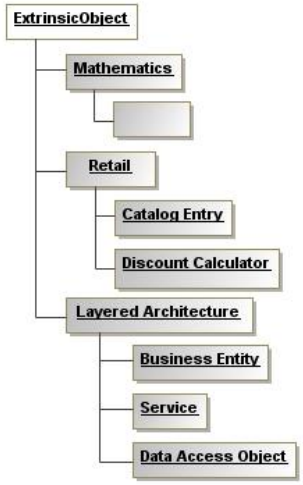

Figure 4. An example of extending ebRIM classification scheme

When storing the components from Figure 3 into the repository, we will extend the description with *Classifier*, instances of *ClassificationNode*, in the following way:

- in class *Product* - Classifier:ClassificationNode[0..*] = ”Catalog Entry”, ”Business Entity”
- in interface *PricingStrategy* and in the component *ProductDiscountCalculator* implementing the interface - Classifier:ClassificationNode[0..*] = ”Discount Calculator”, ”Service”

Thus, searching into the repository can be specified in terms of the classification scheme (see Figure 4), and, for example, the class *Product* will be return as the result of the search when the search keyword is ”Catalog Entry” or ”Business Entity”.

4. Conclusions and future work

The paper is focusing on repository architecture and on the argument that a classification scheme provides an efficient way of organizing the information inside the repository. Consequently the search and retrieve operations will be more efficient.

Analyzing the existing frameworks which supports dynamic availability, and reconfiguration of components, we have proposed a way to adapt and extend such a framework to our purpose: a repository of executable models, with an attached classification scheme.

Further work will concentrate in the near future on two important directions:

- Since the classification scheme that we proposed had only an experimental goal, we intend to align classification criteria with business classification schemes;
- to describe and implement the search and retrieve strategies in the repository.

5. Acknowledgements

This work was supported by the grant ID 546, sponsored by NURC - Romanian National University Research Council (CNCSIS).

References

[1] ebXML *Collaboration-Protocol Profile and Agreement Specification*, http://www.ebxml.org/specfrafts/

[2] Escoffier, C. and Hall R. S., *Dynamically Adaptable Applications with iPOJO Service Components*, In 6th Conference on Software Composition (SC07), pp. 113-128, 2007

[3] Lazăr, I., Pârv, B., Motogna, S., Czibula, I., Czibula G. and Lazăr, L., *iCOMPONENT: A Platform-Independent Component Model for Dynamic Execution Environments*, 10th Internat. Symp. SYNASC, Timisoara, Romania, September, 2008

[4] Mocko, G., Malak, R., Paredis, C. and Peak, R., *A knowledge repository for behavioral models in engineering design*, Proceedings of DETC 2004: 24th Computers and Information Science in Engineering Conference, 2004, 1-10

[5] OASIS, *SCA Service Component Architecture. Assembly Model Specification*, Version 1.1. 2007

[6] OSGi Alliance, *OSGi Service Platform Core Specification*, Release 4, Version 4.1. http://www.osgi.org/, 2007

[7] Pârv, B., Lazăr, I. and Motogna, S., *ComDeValCo framework - the modeling language for procedural paradigm*. International Journal of Computers, Communications & Control (IJCCC), Vol. 3, No. 2, 2008, pp. 183-195

[8] Zaremski, A.M. and Wing, J.M., *Specification matching of software components*, ACM Trans. on Software Engineering and Methodology, **6** (4):333, 1997, 333-369

Babes-Bolyai University
Department of Computer Science
Str Kogalniceanu No 1
400084 Cluj-Napoca, Romania
E-mail address: motogna@cs.ubbcluj.ro
E-mail address: ilazar@cs.ubbcluj.ro
E-mail address: bparv@cs.ubbcluj.ro