The Use of DL in Component Libraries - First Experiences

Ragnhild Van Der Straeten and Miro Casanova
System and Software Engineering Lab
Vrije Universiteit Brussel, Belgium
Email: {rvdstrae|mcasanov}@vub.ac.be

Abstract

Component-based software development (CBSD) is a very promising software engineering technique for improving reuse and maintenance of software components. However, in order to be able to reuse the components, developers should be aware of the available components in the company, in which contexts those components can be used and how they will behave. Our approach is to use the technique of software libraries to classify the components. The structure of these libraries is specified by ontologies. Those ontologies are specified in the $\mathcal{SHIQ}$ description logic. The reasoning services of DL let us verify whether the ontologies are consistent. This helps automating the process of classifying and retrieving a component into a software library.

1 Problem

Component-based software development (CBSD) is one of the major efforts for improving reusability and maintainability of software applications. A component was defined at ECOOP 96 [SP97] as follows: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* The great advantage of component-based software development (CBSD) is that new software can be built by combining bought and self-made components. To do this, it is necessary to specify in which contexts the components can be used and how they will behave.

An important effort for improving reusability of software is the research done on software libraries. A software library is defined as a "managed collection of software assets where assets can be stored, retrieved and browsed" [AM]. One of the main ways of organizing a software library is by using the faceted classification scheme. In this scheme a facet is a "clearly defined, mutually exclusive, and collectively exhaustive aspects, properties or characteristics of a class or specific subject" [T92]. Each facet is defined by a possible hierarchy of terms. Those terms describe the concrete values the facet can have.

We want to classify components based on the description of the semantics of those components. To establish this classification, we use:

- software libraries using the faceted approach.
- ontologies describing the behaviour of the components. These ontologies will represent a multi-dimensional classification of components and will capture the behaviour of components.

We envision this research as a way of improving reuse because of the use of a library which consists of an organized collection of components together with the relevant documentation. We also envision the improvement of evolution management of component-based applications by having the description of those applications in the library, including the dependencies and other relationships between their constituent components. As ontology language the DL $\mathcal{SHIQ}$ [HS00] has been chosen. In the next section we further explain our research and motivate the choice of the logical formalism. In section 3 examples of these ontologies are given. Section 4 concludes this paper.

2 The Framework

It is our intention to classify components in software libraries by using the faceted approach [PD91] [PF87], allowing a multi-dimensional classification of components. We define a dimension as *a set of facets that are related to the same view or the same aspect of a component*. Different dimensions should be considered in the classification. Examples are the functionality (what it does, inputs, outputs, etc.) of a component, the implementation issues (programming language, platform, etc.) and so on. We use facets and terms as defined in [PD91]. To be able to have a multi-dimensional classification of components we describe the different dimensions, their facets and terms. Furthermore, there are also relevant relations that should be described.

We consider two kinds of relations: inter- and intradimensional ones. The first one are relations between dimensions, the latter one are relations within one and the same dimension. These relations put restrictions on the different dimensions, their facets and the respective terms. Remark that this is one way of classifying the kinds of relations. We are working on a more detailed classification taking into account other aspects. To define these structures we will use different ontologies.

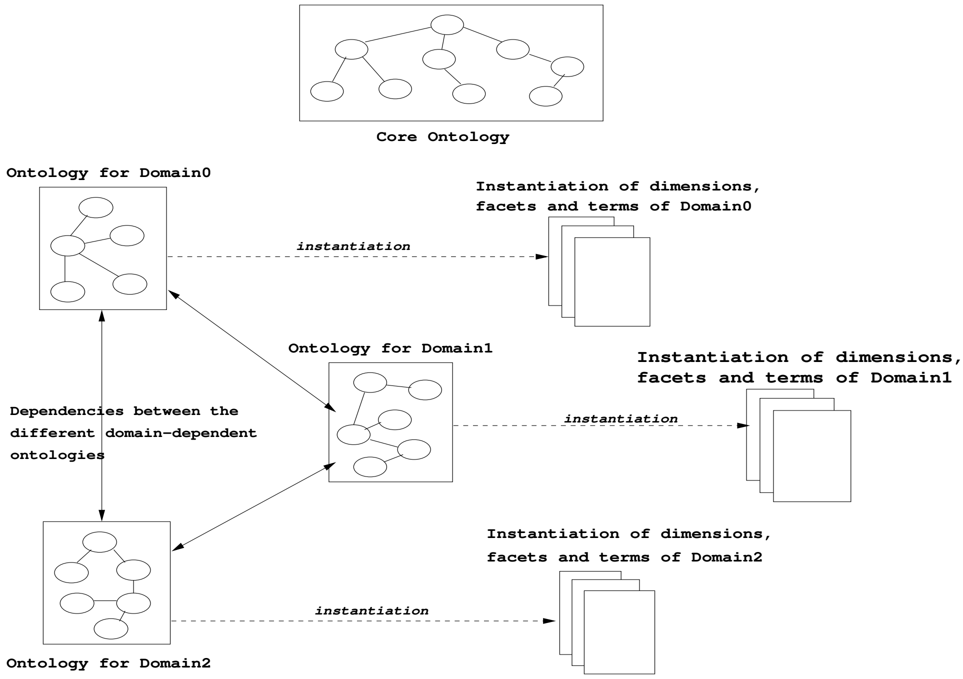

Figure 1: Ontologies for Component Classification

As depicted in figure 1 our framework consists of:

Core Ontology

This ontology consists of dimensions that are relevant for all components. Examples of such dimensions are explained in section 3.1.

Domain-Dependent Ontologies

These ontologies describe the different dimensions relevant for the classification of a given kind of components. Such a kind consist of components belonging to the same domain. Examples of such sets of components are communication network components, user-interface components, data compression components, etc.. Such an ontology will keep track of the facets, terms, dimensions, inter- and intra-dimensional relations of components of one single domain. The advantages of having such ontologies are:

- The knowledge about the components is made explicit by these ontologies.
- The insertion of the components in the library becomes easier because of the explicit presence of the relations. Given a term of a facet, other terms can be automatically derived due to those relations.
- Different libraries based on the same ontology can be easily merged.
- Using ontologies, queries can be executed on software libraries.

As ontology language the very expressive Description Logic $\mathcal{SHIQ}$ [HS00] will be used. The advantages of using Description Logic to build these ontologies are well-known: concepts can be easily composed to form new concepts. DL allows for arbitrary binary relations which enables the expression of the different relations between the components, their terms, facets and dimensions. DL offers efficient reasoning support. This support can be used to reason about and to query the constructed ontology. The specification of these ontologies is terminological expressions.

We remark that next to the inter- and intra-dimensional restrictions within one ontology, we can also have inter-dimensional restrictions between dimensions in different ontologies as indicated in figure 1. We will not consider those relations here because they are still under investigation.

Component Items

They specify the values of the different terms for a specific component. For each component we want to classify, the terms of relevant facets of the different dimensions as described in the ontology are instantiated. Also on this level, we have inter- and intra-dimensional relations. These relations only restrict values concerning these specific component items. Also the instances of the different components and their restrictions will be expressed in DL as assertional knowledge.

In the following section we will give examples of domain-dependent ontologies, component instances and their representation and the reasoning tasks we want to define. The specification of these ontologies and the different dimensions has been figured out by inspecting and analyzing component-based applications and their components. We have to mention that this specification is still evolving.

3 Examples, Representation and Reasoning

3.1 Examples of Dimensions of the Core Ontology

These are dimensions that are relevant for all kinds of components. Two example dimensions are the *implementation* dimension and the *quality of service* dimension. The *implementation* dimension contains information about the implementation of the component such as the platform, the programming language and the compiler (if available). The *quality of service* dimension consists of timing and memory information, or in general it consists of any non-functional requirements or provisions of a component. Its different facets are (so far):

1. Timing provided: a given time-period in which a component will offer a certain service.

2. Timing required: a given time-period within which a component requires some service from another component.

3. Memory required: a given amount of memory a component requires for providing a certain service.

The $\mathcal{SHIQ}$ expressions are shown in figure 2[1].

3.2 Examples of Domain-Dependent Ontologies

An example of such an ontology is the ontology describing the dimensions, facets and terms of network components. In general, the functionality dimension of a component consists of facets that have to do with the functionality part of the component belonging to this domain. In the case of network components, the semantical part of the functionality dimension consists of the following facets and terms:

1. *Kind*: Information about the kind or type of the component.
   - client
   - server

2. *Actions*: the different functions the component can perform. Some terms of this facet are (remark that these lists are not exhaustive):
   - Link
     - Out : reference, subscribe, unsubscribe, connect, disconnect
     - In : referenced, subscribed, unsubscribed, connected, disconnected
   - Data : send, receive, set, get, notify, stream, answer

   An action can have input and output arguments:
   - Input: arguments needed to perform some action.
     - String, Number: {integer, float, ...}, ...
   - Output: results of the performed action.
     - String, Number: {integer, float, ...}, ...

The figure 2 shows a part of this specification in $\mathcal{SHIQ}$.

[1] $\forall^{=1}R.C$ is an abbreviation for $\forall R.C \sqcap (\leq 1R.C) \sqcap (\geq 1R.C)$

3.3 Example of Component Items

Consider a network client component. This component sends strings over a TCP/IP connection and throws events when a string is received or when the connection is established or destroyed. So this component has 4 actions: send, receive, connect and disconnect. The connect action has two input arguments that are both strings (i.e. the port number and the host name), the output of this action is the *rcvConnect* event if the connection succeeds and the *rcvClose* event if the connection fails. This knowledge is specified in Abox assertions.

3.4 Relations and Restrictions

A programming language is related to at least one platform in the implementation dimension. This is an example of an intra-dimensional relation. Another example of such a relation between facets of the implementation dimension, but on the component level is the fact that if the component is compiled using the GNU gcc compiler under Linux, the component's platform must be Linux too. To express this last restriction in DL, we need to be able to express concept inclusion axioms in the Abox as shown in 2. This is possible as explained in [HS01] where $\mathcal{SHIQ}$ is extended with nominals. Consider the network components' ontology again, there exists a relation between the *action* facet of the *functionality* dimension and the *timing provided* facet of the *quality of service* dimension. The *receive* action may take at most 8 seconds. This relation is also written down in figure 2.

3.5 Reasoning

The developer needs the following reasoning services at the ontology level:

1. consistency checking of a domain-dependent ontology,
2. consistency checking of a domain-dependent ontology combined with the core ontology,
3. checking satisfiability of concepts and relations,
4. checking subsumption between concepts or relations.

These reasoning services correspond to the different reasoning tasks of $\mathcal{SHIQ}$. At the component items level, the developer needs support to check the consistency between Abox and Tbox statements and also to check consistency of the Abox. The following remarks can be made concerning the expressiveness needed:

COMPONENT	$\sqsubseteq$	$\forall^{=1}\text{has\_func.FUNCTIONALITY}$ $\sqcap \forall^{=1}$ has_impl.IMPLEMENTATION $\sqcap \forall^{=1}$ has_qos.QUALITYOFSERVICE
IMPLEMENTATION	$\sqsubseteq$	$\forall$has_plat.PLATFORM $\sqcap$ $\forall$has_lang.LANGUAGE $\sqcap$ $\forall$ has_comp.COMPILER
QUALITYOFSERVICE	$\sqsubseteq$	$\exists$timeprovided.$\top$ $\sqcap$ $\exists$timerequired.$\top$ $\sqcap$ $\exists$ memoryrequired.$\top$
FUNCTIONALITY	$\sqsubseteq$	$\forall^{=1}$has_act.ACTION $\sqcap$ $\forall^{=1}$has_kind.KIND
CLIENT	$\sqsubseteq$	KIND
SERVER	$\sqsubseteq$	KIND
ACTION	$\sqsubseteq$	$\forall$has_output.OUTPUT $\sqcap$ $\forall$has_input.INPUT
DATA	$\sqsubseteq$	ACTION
SEND	$\sqsubseteq$	DATA
RECEIVE	$\sqsubseteq$	DATA
IMPLEMENTATION $\sqcap$(= 1 has_lang.LANGUAGE)	$\sqsubseteq$	($\geq$ 1 has_plat.PLATFORM)
COMPONENT $\sqcap$ $\exists$has_func.(FUNCTIONALITY $\sqcap$ $\exists$has_act.RECEIVE)	$\sqsubseteq$	$\exists$has_qos.(QUALITYOFSERVICE $\sqcap$ $\exists$($\leq_8$ timerequired))
gcc-Gnu-Linux	:	COMPILER
Linux	:	PLATFORM
$\exists$has_comp.gcc-Gnu-Linux	$\sqsubseteq$	$\exists$has_plat.Linux $\sqcap$ ($\leq$ 1 has_plat)

Figure 2: Definitions of Ontology Concepts and Relations.

- First of all, we need concrete domains to express certain relations. For example, the input arguments of an action can be integers or strings. Unary concrete domains will not be sufficient in this case. In [CL02], $\mathcal{Q}-\mathcal{SHIQ}$ is introduced, which extends $\mathcal{SHIQ}$ with concrete domain concept constructors allowing the representation of rational numbers. However in our case, we need to consider Integer, String and real numbers and it should also be possible to express intervals of real numbers. For example the time required by a certain offered service can be between two real values. The question is of course if decidability is lost if $\mathcal{Q}-\mathcal{SHIQ}$ is further extended.

- It should also be possible to use instances of Tbox concepts, i.e. Abox individuals in Tbox expressions. The name of a certain component can appear in the domain-dependent ontology to which this component can be classified. So we need nominals in our language. In [HS01], $\mathcal{SHIQ}$ is extended with nominals.

4 Conclusion

In this paper we presented our ideas to classify software components in libraries based on ontologies described in the very expressive DL $\mathcal{SHIQ}$. We use Description Logics to specify different ontologies used to classify components. The reasoning services of DL let us verify whether the ontologies are consistent, whether dimensions, facets and terms are satisfiable, whether certain relations or certain concepts in the ontology (i.e. dimensions, facets and terms) subsume each other. This helps automating the process of classifying and retrieving a component into a software library. The next step in this research is to make a more detailed classification of the different possible relations appearing in the ontologies. This will allow us to find out if existing DLs ($\mathcal{SHIQ}$ in particular) are expressive enough for our purposes. Another step is to use a $\mathcal{SHIQ}$ reasoner allowing us to reason about the different ontologies and component items. Finally, we will define queries about the component items in order to retrieve components or specific facets or terms.

References

[SP97] Szyperski, C., & Pfister, C. (1997). Workshop on Component-Oriented Programming, Summary. In MuhlHauser M. (Ed.) Special Issues in Object-Oriented Programming - ECOOP96 Workshop Reader. Dpunkt Verlag, Heidelberg.

[BJ99] Beugnard, J. Jezequel, N. Plouzeau, D. Watkins. "Making Components Contracts Aware". IEEE Computer, July 1999.

[PD91] R. Prieto-Diaz. "Implementing Faceted classification for software reuse". Communications of the Acm. May 1991.

[PF87] R. Prieto-Diaz and Peter Freeman. "Classifying software for reuse". IEEE Software, 4(1):6-16, January 1987.

[AM] S. Atkinson and A. Mili. "Software Libraries", *http://citeseer.nj.nec.com/17413.html*.

[T92] Taylor A. "Introduction to Cataloging and Classification". 8th ed. Englewood, Colorado: Libraries Unlimited, 1992.

[HS01] I. Horrocks and U. Sattler. Ontology Reasoning in the SHOQ(D) Description Logic. In the Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, 2001. Seattle, USA, 2001.

[HS00] I. Horrocks, U. Sattler and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In the Proceedings of CADE-00, pg. 482-496, 2000.

[CL02] C. Lutz. Adding numbers to the $\mathcal{SHIQ}$ Description Logic - First Results. In the Proceedings of the Eight International Conference on Principles of Knowledge Representation and Reasoning (KR2002). Morgan Kaufman. Toulouse, France, 2002.