

KOntoR: An Ontology-enabled Approach to Software Reuse

Hans-Jörg Happel*), Axel Korthaus†), Stefan Seedorf†) and Peter Tomczyk*)

*) *FZI Research Center for Information Technologies
Research Group Information Process Engineering (IPE)
Haid-und-Neu-Str. 10-14
D-76137 Karlsruhe, Germany
{happel|tomczyk}@fzi.de*

†) *University of Mannheim
Lehrstuhl für Wirtschaftsinformatik III
Schloss, L 5,5
D-68131 Mannheim, Germany
{korthaus|seedorf}@wifo.uni-mannheim.de*

Abstract

Research on software reuse libraries has extensively dealt with representation and retrieval issues of software artifacts. While representation in terms of metadata is a key issue, most systems neglect the possibilities of leveraging knowledge about the corresponding problem domain. In this paper, we present KOntoR—an ontology-enabled approach to software reuse. We show how background knowledge provided in the form of ontologies can increase the value of reuse libraries. This is achieved by integrating explicit and implicit metadata semantically, thus providing means for deriving new facts. Additionally, we give three examples that show how software library users can benefit from formalized knowledge, e.g. about software licenses or programming technologies.

1. Introduction

Although the topic of reuse has been widely discussed in the area of software engineering for many years, many researchers and practitioners are not yet satisfied with the current state of practice [1]. Component libraries, service registries or artifact repositories constitute an indispensable part of software reuse systems. They all have in common that they manage artifact descriptions for supporting the process of retrieval, adaptation and integration. However, current approaches often fall short of providing adequate support

for typical reuse tasks. The paper thus particularly addresses the following problems:

- In many cases, the relevant information resides isolated in multiple heterogeneous descriptions of an artifact, covering different aspects each. It should therefore be examined how these descriptions can be integrated.
- A purely metadata-based approach is often not powerful enough to answer declarative queries (see Q.1 - Q.3) and thus requires additional background knowledge. To date, most reuse systems do not incorporate formalized knowledge about application domains or artifacts.

We argue that semantic web technologies provide the means for addressing these issues. In this paper, we present *KOntoR*, an infrastructure for software reuse employing technologies from the emerging field of semantic web research. We show that codifying knowledge in an ontology-enabled infrastructure can significantly improve the value of a reuse environment.

A typical usage scenario of a reuse environment would include the retrieval of particular software components fitting a specific application development need. However, realistic reuse scenarios require a wider range of supported functionality. The multifaceted requirements on such enhanced software reuse systems can be illustrated in the form of competency questions (cf. [2]).

These are example queries that should be supported by our ontology-enabled approach, such as:

- Q.1 Find all artifacts dealing with the *customer* business object.
- Q.2 Tell me if componentA and componentB can be used in my product under a proprietary license.
- Q.3 Find a developer who is experienced in building banking applications in Java.

The paper is structured as follows: We first analyze the shortcomings of current reuse approaches. In chapter 3, the architecture of the KOntoR approach is derived. Chapter 4 describes the prototype implementation and gives an example for the realization of Q.1 – Q.3, before summarizing the key advantages of our approach.

2. Related work

Approaches to software reuse target different kinds of reuse artifacts—ranging from classical binary, *off-the-shelf* components to source code fragments, process knowledge and *experience* [3, 4, 5]. However, to manage and retrieve these artifacts, an appropriate infrastructure has to be in place. In this chapter we will therefore review metadata-oriented and intelligent reuse systems.

2.1 Metadata-based reuse

Unlike text documents, most reuse artifacts in software engineering are not human-readable. This applies especially to binary code, but also the serializations of design models or test cases are not necessarily understandable in their raw form [6]. Also, access might be unfeasible due to the artifact's complexity. Thus, some representation of an artifact is needed that can be matched against a user's request [7, 8]. Since this representation is describing the actual reuse artifact's data, we call it metadata. A general setting of software reuse would therefore involve three essential success factors as depicted in Figure 1.

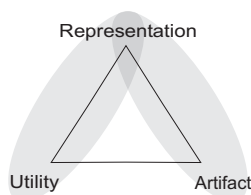


Figure 1: Dimensions of reuse (derived from [6])

For the user to maximize utility a library containing suitable artifacts and an appropriate representation in the form of queryable metadata describing those arti-

facts is required [6]. Traditional libraries for reuse artifacts serve as an intermediary between component providers and component requesters. Accordingly, they have two clearly defined interfaces to the software development process—providers use the repository once when adding new components and buyers access the repository to search for components. As a result, component libraries focus on effective retrieval mechanisms and are based on so-called descriptive metadata. For example, components can be classified by using a fixed set of facets [9] or fixed taxonomies, both assuming a stable body of knowledge. In this sense, software is regarded as a kind of document—like in *library science*—that can be handled by classification and indexing methods. Thus, most reuse libraries allow for different kinds of artifacts, but are mostly limited to a fixed number of representation formats and users [6, 10].

In contrast to component libraries, collaborative development platforms aim at supporting distributed development teams. They offer different features, e.g. bug-tracking, communication tools, project management or version control. Although some platforms include basic project and artifact descriptions, they deal with administrative metadata in the first place. While primarily designed for supporting software development, sites like *SourceForge*¹ have become popular places to search for reusable components. However, metadata and retrieval mechanisms are less powerful than in component libraries. SourceForge for example is limited to keyword based search and browsing a simple classification taxonomy (TROVE-scheme²).

While metadata-based reuse has shown some success—especially in intra-organizational settings [11] and for infrastructure tools [3]—many authors claim that it has failed a breakthrough [1, 3, 11]. Especially in distributed development settings spanning multiple sites or time zones, a common conceptualization of development tasks and some background knowledge is required [12], which the presented tools do not provide.

2.2 Intelligent reuse systems

The rising number of artifacts, metadata formats and actors in modern software engineering processes imposes additional requirements on the design of reuse systems. Vitharana et al. [10] tackle this by differentiating among several user groups and representations of

¹ <http://sourceforge.org>

² <http://sourceforge.net/softwaremap/index.php>

structured and semi-structured data, as well as providing basic support for inferences. However, additional knowledge about the semi-structured data or its querying possibilities is not provided. Intelligent reuse systems try to remedy this by formally capturing context of an artifact or the user. Together with background knowledge, more powerful reuse results can be obtained. A knowledge-based reuse infrastructure providing personalized views for different stakeholders with different needs can add value to all three dimensions depicted in Figure 1.

Within the *utility* dimension, the structuring and capturing of user context can inform the reuse system. Recommender systems like Hipikat [13] or CodeBroker [14] gain information about the context of a developer or his task and recommend information that seems to be suitable in this context. However, these tools are integrated in a development environment (Eclipse resp. Emacs) and they are primarily designed to work in an intra-project setting. Additionally, the knowledge used for recommendation is not declaratively stored, but hard-coded.

Development information systems aim to enhance the *representation* of artifacts. Tools like the LaSSIE system [15] use description logic formalisms to store knowledge about the application domain and code structure into a knowledge base [16]. Thus, the system is able to answer declarative requests about the code using links between the domain model and the source code. Falbo et al. [17] work towards an ontology-based software development environment (ODE). Their vision includes the entire software development lifecycle by capturing process, tool and quality in ontologies.

Background knowledge can also be leveraged in the *artifact* dimension. Oberle [18] proposes an ontology-enabled application server for managing distributed systems. Here, semantic technologies are also used to describe relationships between artifacts in an application server, thus enabling use-cases such as the automatic discovery and loading of dependent libraries at run-time. While this system focuses on run-time application management instead of reuse, it clearly demonstrates the benefit of combining ontologies with existing metadata. Thus, the approach proposed by Oberle addresses both the representation dimension and the artifact dimension of the reuse triangle in Figure 1.

We think that combining “intelligent” approaches with the features of metadata-based reuse systems may lead to a powerful solution. Such an integrated approach is proposed and described in detail below.

3. The KOnToR solution approach

The identified shortcomings of existing software reuse systems, namely the low integration of artifact metadata and insufficient utilization of background knowledge, are addressed by two key elements in our architecture (see Figure 2):

1. An XML-based *metadata repository component* to describe software artifacts independently from a particular format
2. A *knowledge component* which comprises an ontology infrastructure and reasoning to leverage background knowledge about the artifacts

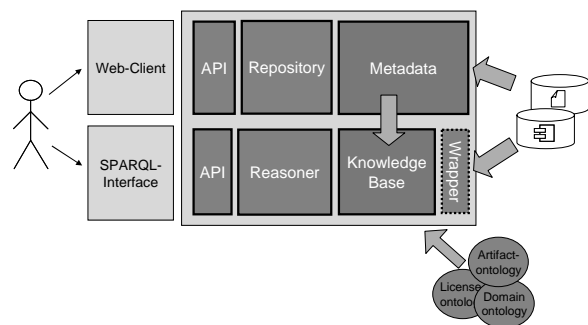


Figure 2: KOnToR high level architecture

In order to allow for the management of arbitrary metadata, the repository component is based on a particular metaschema (see Figure 3). Every metadata set is an XML document which describes a software artifact in one or more information aspects. For example, a WSDL³ document describes how to communicate with a component’s service and covers the information aspect *Interface*. The Dublin Core standard⁴ may be used to specify the aspects *Authorship* and *Licensing* of a component.

Furthermore, different formats may be applied to represent an information aspect of a software artifact. This feature is also covered by the proposed metaschema. For example, the interface of a software component may be described using WSDL, Corba IDL or a UML profile serialized in XMI. For being able to fulfill the requirements stated above, the artifact metadata needs to be automatically homogenized, e.g. by transforming a WSDL definition into a simplified, consistent interface description, thereby unifying different formats.

³ <http://www.w3.org/2002/ws/desc/>

⁴ <http://dublincore.org/>

The artifact metadata is managed using a dedicated API. Artifact retrieval is supported by a SQL-style XML-based query language (cf. [19]). It can be used for creating structured queries (via XPath expressions⁵) as well as for keyword-based search.

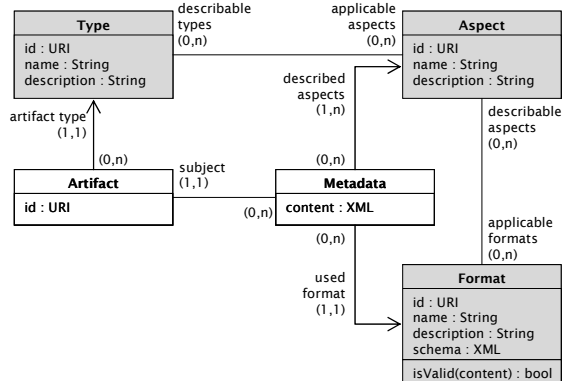


Figure 3: Metaschema

The second building block of the KOnToR architecture, the knowledge component, comprises a knowledge base which imports the transformed metadata. This is achieved by writing a mapping ontology and an export plug-in for each data source. In this way, assertional knowledge can be combined with background knowledge provided by ontologies. In the center, there is a “shallow” ontology for describing structural aspects of software artifacts (see Figure 4). This ontology is extended by optional domain ontologies, e.g. an ontology for describing standard software licenses. The background knowledge contained within the domain ontologies can be utilized to assist the user in a wide range of reuse problems. In the process of component selection, a system integrator might want to know whether an open source component (e.g. derived from GPL) can be reused in a commercial project under a proprietary license. The proposed architecture supports such queries since the necessary background knowledge for testing the compatibility of free and proprietary licenses is provided by a domain ontology which can be plugged in easily.

Semantic queries on top of the knowledge base are supported by a different API. The functionality of this interface concentrates on processing SPARQL⁶ queries executed by an interpreter and returning the results. Finally, simple clients for both APIs are required to provide end users access to the system.

4. Implementation and example

The architecture presented in the preceding section has been implemented in the context of the CollaBaWue⁷ project. We will first describe selected implementation aspects and then give a concrete example to demonstrate how the identified use cases are realized.

4.1 Prototype implementation

The two main components of the KOnToR architecture have been realized as a prototype in Java. The repository component provides an API for managing artifact metadata and an instantiation of the metaschema. The metadata is stored in a Hypersonic relational database (hsqldb)⁸ and indexed with Lucene, by using individual analyzers (e.g. one for XMI). Moreover, a web interface makes it possible to manage artifact descriptions.

For the prototype implementation of the knowledge component, we set up an RDFS/OWL⁹ based infrastructure. The KAON2 API¹⁰ was used for both storing the ontologies and reasoning. Declarative requests are supported using a simple front-end for KAON2’s SPARQL interface.

In order to explain the example in the subsequent section, we need to derive an instantiation of the metaschema and specify ontologies for providing background knowledge. Table 1 shows the excerpt of a metaschema instance which can be extended to describe various aspects of software components. The metadata is specified by using standardized (e.g. WSDL, Dublin Core) as well as custom XML-based formats.

| Type | Component |
|---------------------|--|
| Information aspects | Interface, Behavior, License, Authorship |
| Formats | WSDL, XMI, DublinCore |

Table 1: Instantiation of the metaschema

For our example presented below, we define an artifact ontology and additional domain ontologies, i.e. a banking, technology and software license ontology, which are part of the prototype’s knowledge component.

⁷ <http://www.collabawue.de>

⁸ <http://hsqldb.org/>

⁹ <http://www.w3.org/2001/sw/>

¹⁰ <http://kaon2.semanticweb.org/>

⁵ <http://www.w3.org/TR/xpath>

⁶ <http://www.w3.org/TR/rdf-sparql-query/>

4.2 Example

In order to demonstrate the applicability of our approach, we introduce the example of a banking component (*AccountBalance*) which returns the balance of a private customer's account. The component is realized as Web service and comprises a WSDL description of the component interface. In the excerpt below, a domain reference of the input element to the *PrivateCustomer* business object is established following an annotation approach comparable to the WSDL-S proposal (cf. [20]):

```
<wsdl:description ...>
<wsdl:types>
<xs:schema ...>
  <xs:element name="c" type="Customer"
    bo:businessObjectRef="fin#PrivateCustomer">
    ...
  </xs:element>
  ...
</xs:schema>
</wsdl:types>
<wsdl:interface name="GetAccountBalance">
<wsdl:operation name="getBalance"
  pattern="http://www.w3.org/.../wsdl/in-out">
  <wsdl:input element="my:Customer"/>
  <wsdl:output element="my:Balance" />
</wsdl:operation>
</wsdl:interface>
</wsdl:description>
```

Further, we specify that the license held on the component is GPL. To this end, we adopt the Dublin Core rights management element. The following XML document describes the component authorship and uses a qualified name to reference a concept in our license ontology (*lic:GPL*), which is a taxonomy of software licenses.

```
<metadata xmlns=...>
  <dc:rights> lic:GPL </dc:rights>
  <dc:creator> Dave Developer </dc:creator>
</metadata>
```

These XML-based artifacts belonging to the *AccountBalance* component are described according to the metaschema instance and stored in the repository using the repository component's API.

Subsequently, these artifacts can be extracted, provided that a format wrapper has been defined, and inserted into the knowledge base. The ontologies together with the concepts used in our example are visualized in Figure 4. The artifact ontology provides the skeleton to describe components and related artifacts. It is supported by three domain ontologies providing the background knowledge for realizing the three use cases. As an example, the finance ontology (*fin:.*) defines that *PrivateCustomer* is subsumed by *Customer*.

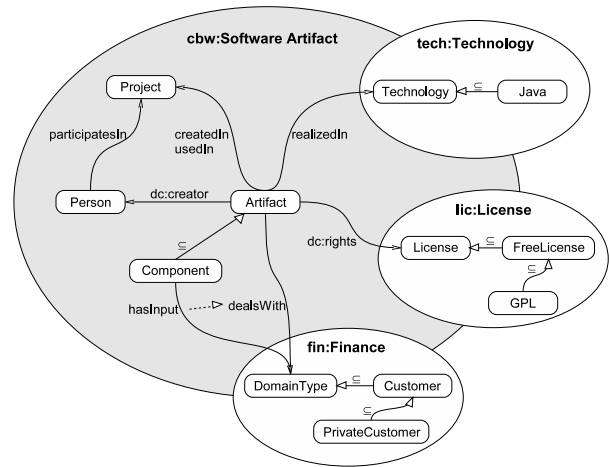


Figure 4: Part of the KOnToR ontology

The knowledge component can be applied to define a wide range of queries as required by the competency questions Q.1-Q.3 formulated in section 1. The following SPARQL query returns all artifacts dealing with the *Customer* business object as postulated in use case scenario Q.1:

```
SELECT ?cbw:artifact WHERE
{?cbw:artifact <cbw:dealsWith>
<fin:customer>}
```

In this query, the *AccountBalance* component, which uses an annotation to specify *PrivateCustomer* as its input parameter, is returned. This is possible since the knowledge base is aware of *AccountBalance* being an *Artifact*, *PrivateCustomer* being a specialization of *Customer* and *hasInput* being a subproperty of *dealsWith*.

According to example Q.2, we specify another query in order to verify if component A and component B can be used in the product to be developed under a proprietary license:

```
SELECT ?compA, ?compB WHERE {
  ?compA <dc:rights> ?lic1;
  ?compB <dc:rights> ?lic2;
  ?lic1 <lic:isCompatible> ?lic2}
```

The third example query Q.3 is about finding a developer who is experienced in building banking applications in Java:

```
SELECT ?developer WHERE {
  ?developer <dc:creator> ?comp;
  ?comp <cbw:realizedIn> <tech:Java>;
  ?comp <cbw:dealsWith> <fin:DomainType>}
```

The information that the component is realized in Java can be extracted from the enclosed binary package not described in this example.

5. Summary and conclusion

In this paper, we addressed two major problems underlying current software reuse systems: the low integration of reusable assets and the insufficient utilization of knowledge about the artifact's domain. The presented KOntoR architecture and prototype implementation takes a two-step approach for an evolutionary adoption in software processes. First, we contributed a repository component which achieves the requested integration of artifact metadata. Second, the knowledge component uses ontologies to capture the background knowledge required for declarative queries. As shown in the examples, the system is capable of supporting a wide range of tasks that are currently not supported by other reuse systems.

For future work we plan to develop additional user interfaces for different interactions with the repository. The options under consideration are a semantic wiki and a context extraction module for an active repository (cf. [Ye01]). Moreover, there are some open tasks in the core of the system. The interplay between artifact metadata, knowledge base and external data sources should be generalized. Not only does this include a mechanism for mappings between equal concepts and instances with different representations but also an efficient strategy for fetching just the required metadata from the data sources when a declarative query is executed. Finally, we are planning to evaluate the system in a real-world environment within the scope of the CollaBaWue project.

6. References

- [1] W.B. Frakes and K. Kang, "Software Reuse Research: Status and Future", in: IEEE Trans. on Softw. Eng., vol 31, no. 7, 2005, pp. 529-536.
- [2] M. Uschold and M. Gruninger, "Ontologies: principles, methods, and applications", Knowledge Engineering Review, vol. 11, 1996, pp. 93-155..
- [3] C. Szyperski, "Component Software - Beyond Object-Oriented Programming". Addison-Wesley, 2nd ed., London, 2002.
- [4] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise", in Proc. of the 24th Int. Conf. on Software Engineering (ICSE), Orlando 2002, pp. 503-512.
- [5] V. R. Basili and G. Caldiera, "Improve Software Quality by Reusing Knowledge and Experience", Sloan Management Review, 1995, pp. 55-64.
- [6] A. Mili, R. Milli and R.T.: Mittermeir, "A survey of software reuse libraries", in: Annals of Software Engineering, vol. 5, 1998, pp. 349-414.
- [7] T. J. Biggerstaff and C. Richter, "Reusability framework, assessment, and directions", in Software reusability: vol. 1, concepts and models: ACM Press, 1989, pp. 1-17.
- [8] C.W. Krueger, "Software Reuse", ACM Comp. Surv., vol. 24, 1992, pp. 131-183.
- [9] R. Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", in: Comm. ACM, vol. 34, no. 5, 1991, pp. 88-97.
- [10] P. Vitharana, F. Zahedi and H. Jain, "Knowledge-Based Repository Scheme for Storing and Retrieving Business Components", in: IEEE Trans. on Softw. Eng., vol. 29, no. 8, 2003, pp. 649-664.
- [11] J. Bosch, "Software Product Lines: Organizational Alternatives", In Proc. of the 23rd Int. Conf. on Software Engineering (ICSE), Toronto, 2001, pp 91-100.
- [12] J. A. Espinosa, R. E. Kraut, J. F. Lerch, S. Slaughter, J. D. Herbsleb and A. Mockus: "Shared Mental Models and Coordination in Large-Scale, Distributed Software Development", In Proc. of the Int. Conf. in Information Systems (ICIS), New Orleans, 2001, pp. 513-518.
- [13] D. Cubranic, G.C. Murphy, J. Singer, K. S. Booth, "Hipikat: A Project Memory for Software Development", in: IEEE Trans. on Softw. Eng. vol 31, no. 6, 2005, pp. 446-465.
- [14] Y. Ye and G. Fischer, "Reuse-Conducive Development Environments", Autom. Softw. Eng. vol 12, no. 2, 2005, pp. 199-235.
- [15] P. T. Devanbu, R. J. Brachman, P. G. Selfridge and B. W. Ballard, "LaSSIE: A Knowledge-Based Software Information System", in Comm. ACM, vol 34, no. 5, 1991, pp. 34-49.
- [16] C.A. Welty, "Software Engineering", in: Description Logic Handbook, 2003, pp. 373-387.
- [17] R.A. Falbo, D. O. Arantes and A.C.C. Natali, "Integrating Knowledge Management and Groupware in a Software Development Environment", In Proc. of the 5th Int. Conf. on Practical Aspects of Knowledge Management (PAKM), Vienna, 2004, pp. 94-105.
- [18] D. Oberle, "Semantic Management of Middleware", Springer, New York, February 2006.
- [19] H.-J. Happel, A. Korthaus, S. Seedorf and P. Tomczyk, "Ein Ansatz zur formatneutralen Verwaltung von Metadaten in komponentenorientierten Softwareprozessen", in: Proc. of Software Engineering 2006, Leipzig, March 28-31, 2006, pp 181-192.
- [20] R. Akkiraju et al.: "Web Service Semantics - WSDL-S", W3C Member Submission, 7 Nov., 2005.