Selection of Middleware-oriented Styles with Ontology-based Metadata

Reiner Jung[1] and Simon Giesecke[2]

[1] Carl von Ossietzky University, 26111 Oldenburg, Germany
[2] OFFIS Institute for Information Technology, Escherweg 2, 26121 Oldenburg, Germany

Abstract. The selection of an appropriate architectural style is an important task in the development of a complex software system. In practice, the decision for an architectural style is often made ad-hoc and not in a systematic way. We present an approach for modelling architectural style metadata. It is based on extracting properties of middleware-oriented architectural styles from the documentation. The metadata is represented using a OWL/SWRL ontology. In addition, we present an approach for using this metadata to support the selection of an architectural style. A case study for a Web-based information system evaluates the proposed approach.

1 Introduction

Software systems and especially middleware-based enterprise application systems have grown in size and complexity. To control and manage these complex projects, new methods, techniques and processes have been developed. One improvement for software development was the use of architectures and architectural patterns and formally specified styles.

While there are only a few generic styles known today, platform induced styles [1] for middleware-based systems are connected to a vast number of technologies and combinations of technologies, and therefore the number of middleware-oriented styles could be enormous.

Many of these middleware-oriented styles have similar properties, restrictions, and target domains. To find the appropriate style in a collection of similar styles for a given project is a complex and costly effort, because each style has to be evaluated in respect to the requirements for the specific project.

To reduce the effort, the styles could be arranged in a style taxonomy, where the most generic style is the root of a branch of styles and with each level down, the styles get more specialized. To select a style in such taxonomy, a hierarchical approach for the selection of candidate architectural styles can be applied.

However, such an approach has three potential shortcomings. First, the selection has to be done by an engineer, because for each node in the taxonomy the engineer has to make a value judgement or decision. This is a time consuming process. Second, the use of a taxonomy reduces the style relations into a tree-like structure, so multiple inheritance in styles are not possible. Thirdly, while hierarchically examining styles, style candidates might be ignored. For example a suitable sub-style in one branch is left out because the root style of the branch is considered less valuable than one of its siblings.

The herein introduced ontology-based approach addresses these three shortcomings. First, the hierarchical approach is replaced by a computer based search, where the project requirements form the search criteria. Second, the information storage is ontology- and rule-based, which allows more complex relationship structures. Furthermore ontology- and rule-based systems allow the use of reasoners to infer new knowledge and solvers to find solutions on a logical rather than a sub-string basis. And three, a solver for a description logic based ontology can find all solutions to a given query, because it checks all styles.

As basis for this article, semantic web technologies are used. This allows different people at different places to input data. Also these technologies are supported by a large variety of software systems, which can be used to build and search the knowledge base, as well as do reasoning with the knowledge base.

In section 2 we discuss the basic concepts which ensure a common foundation for the knowledge base. On this foundation, knowledge of middleware-based styles can be collected and stored in the knowledge base with the technique explained in section 3. Section 4 introduces a querying technique which allows to find styles in the knowledge base. In section 5 the two techniques are illustrated with a case study based on Bornhold's thesis [2]. And section 6 discusses the benefits and shortcomings of this approach and addresses potential future improvements and uses for the two techniques.

2 Knowledge Base Model

The *Knowledge Base Model* (KBM) introduces a basic set of concepts and properties to describe the founding element of architectural styles as well as a set of concepts and properties for the domain model of middleware styles. These concepts and properties ensure cohesion in the knowledge base, because they connect different styles with the common vocabulary they represent.

Fig. 1 shows the two levels of the ontology-based knowledge base. The upper level illustrates individuals, like the Spring style, and the lower level represents the concepts of the knowledge base. The gray circles show how the individuals and concepts are related to each other.

2.1 Basic Style Concepts, Properties and Rules

In [3] an ontology-based architectural style modeling approach is introduced which is our source for the basic style concepts of the KBM. However, the available solver package [4] for OWL DL [5] and SWRL [6] cannot handle the OWL property *subclassOf* as rule predicate and classes as objects (the later is a limitation of OWL DL and SWRL). Therefore our approach remodels the inheritance property with a new property *hasParent*. This has the disadvantage, that styles cannot be modeled as concepts but as individuals. And therefore the inheritance

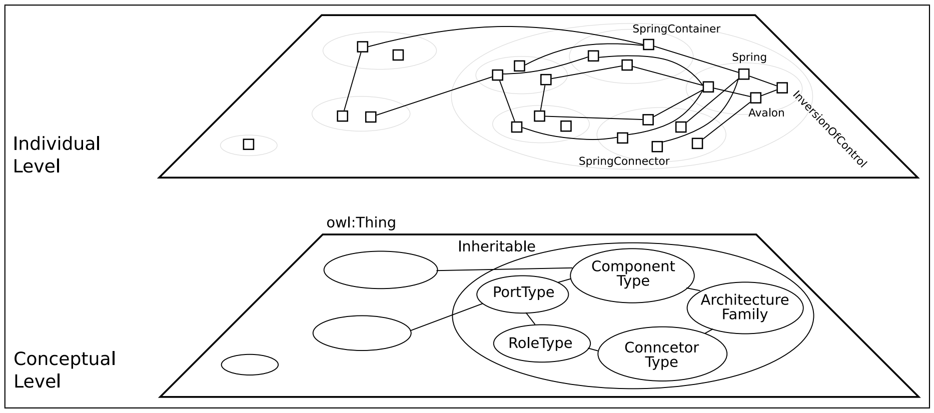


Fig. 1. Illustration of the individual and conceptual levels in a knowledge base and their relation to each other.

of properties has to be modeled with SWRL rules. However, the benefit of using individuals to model styles is, that SWRL-based queries can be used, which is required by the query technique (see section 4).

The primary concept for architectural styles in our work is *Architecture-Family*. The most important subclass of *ArchitectureFamily* is *MiddlewareOrientedArchitecutureFamily* which represents the subset of middleware oriented architectural styles. Furthermore it is a subclass of *Technology*, and therefore it inherits the properties of the technology concept.

The further concepts for styles are component, connector, port and role type [3] which are the composing elements or a style. The association between the style and its component and connector types is realized through the *hasComponentType* and *hasConnectorType* property. The connection between components and connectors is modeled with ports and roles. Therefore component and connector types have port and role types, respectively. The association of these types is realized with *hasPortType* and *hasRoleType*. Beside these associations, the port and role types are related to each other with *acceptsRole* and *connectsTo* properties. The *connectsTo* property is defined as the inverse of *acceptsRole*, so relational information has only to be defined for one property and it becomes available to the other.

Beside these properties used to model architecture families, relationships of these families and their founding elements is also necessary to improve search capablities. All relationship properties are based on the existing *hasParent* property. The relationship properties are *hasRelative* which describes the broader relationship of elements and families, *hasAncestor* which describes the relationship of an element to all its ancestors, *hasDescendant* is expresses the other direction, which means all children and children of children are in this relation, and *hasSibling* implements the sibling relationship.

2.2 Basic Domain Model

The basic domain of middleware-oriented architectures comprises technology, features, pattern, licenses, programming languages and common elements of the target domains. For example a technology for web-applications has properties unique to the domain of web-applications.

On the basis of [7] and the case study in [2] we developed a set of common concepts for the knowledge base. Below a sub-set of these concepts is introduced:

Technology represents the concept of a concrete technology implementation like Spring, Avalon, Orbit, or GStreamer.

Package is a part or subset of functions and components of a technology. For example the Spring Framework comprises of several parts for data handling, for component embedding, etc.

Feature is the ability of a technology, a package, a component, or a connector to perform or act in a certain way. For example XML-processing or streaming.

ProgrammingLanguage represents the concept of a concrete programming language, like fortran77 or c92.

Standard is the concept representing specifications for technologies, programming languages, document formats etc. For example CORBA is a standard and Orbit is an implementation of it.

License is the concept to represent the various licenses used in the software domain.

DesignPattern represents the concept of a general reusable solution to a common problem without a formal specification.

Topology represents concept describing the different ways the components and connectors are combined in an architecture. In [7], the distinguished topologies are: acyclic, arbitrary, hierarchy, incomplete graph, linear, and star.

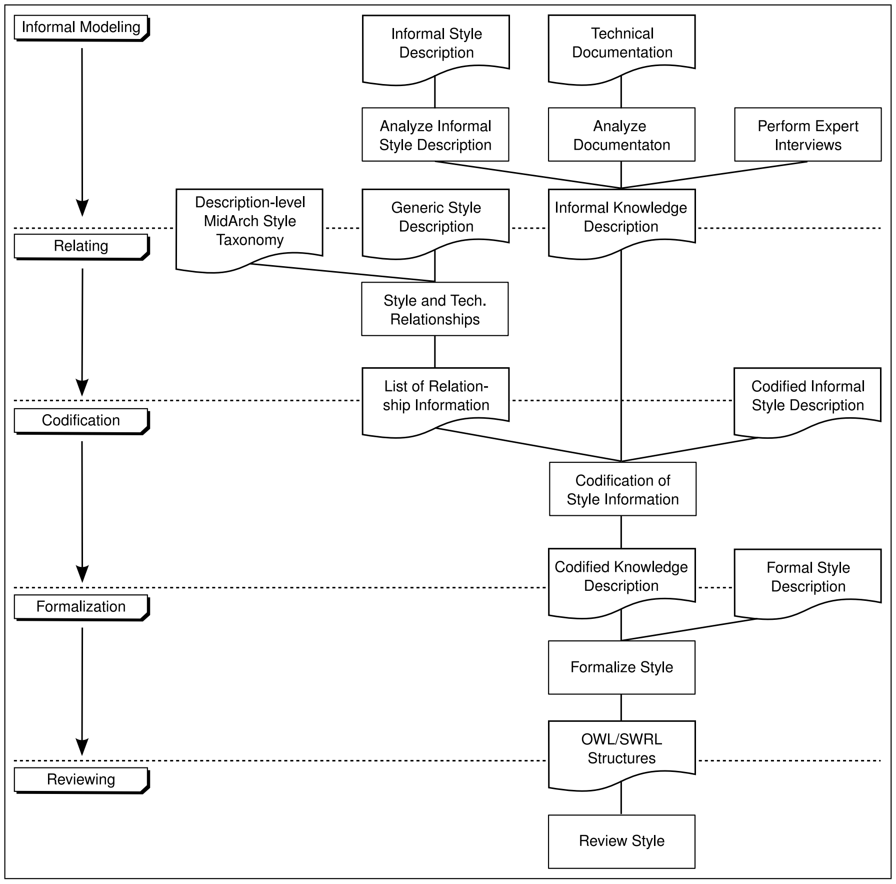
3 Knowledge Modeling Technique

The *knowledge modeling technique* (KMT) is used to add meta-information to the knowledge base in a structured way. The technique can easily be supported and even automated by tools. As the KMT is a knowledge retrieval technique in the context of the MidArch method [8], it is modeled closely to the MidArch style modeling process. The KMT introduces five steps: informal, relating, codification, formalization and review (see Fig. 2).

3.1 Informal Modeling

The *informal modeling* task is used to introduce new information to the knowledge base. The import sources for the knowledge retrieving process are the informal style descriptions, the technical documentation of the involved technologies, and the results of expert interviews on the technology and architectural style.

The informal style description provide basic component, connector, port, and role type description, some general information on the style, and reference to



Informal Modeling
Informal Style Description
Technical Documentation
Analyze Informal Style Description
Analyze Documentaton
Perform Expert Interviews
Description-level MidArch Style Taxonomy
Generic Style Description
Informal Knowledge Description
Relating
Style and Tech. Relationships
List of Relation-ship Information
Codified Informal Style Description
Codification
Codification of Style Information
Codified Knowledge Description
Formal Style Description
Formalization
Formalize Style
OWL/SWRL Structures
Reviewing
Review Style

Fig. 2. The knowledge modeling procedure, its tasks, and artifacts

technologies. The technical documentation contains in most cases information on the target domain of the technology, the parts, packages, components and connectors of the technology, and all features of the technology. And the optional interview depends on the availability of a platform expert. The platform expert can provide further insights to the technology and act as an early reviewer of the informal information, gathered in the analysis of the informal style description and the technical documentation.

In the following we explain the knowledge aspects which are used as major selection criteria for document phrases, and analysis methods used for the input sources.

Knowledge Aspects. The first aspect covers the *target domain* of the analyzed platform and technology, as well as the goals the platform developers wanted to achieve. The second aspect focuses on technical information, information like supported design pattern, used technologies, features, and related frameworks. This aspect is the *technical aspect*. The third aspect relates to quality attributes, like interoperability, stability, and documentation quality. It is therefore called *quality aspect*.

Documentation Analysis Methods. The informal style description and the technical documentation are text documents. The information in these documents must be extracted in a structured way, so that the information can be further processed and finally stored in the knowledge base.

The informal style description contains mainly statements describing the architecture family in an already structured way. These statements can be extracted and categorized by the three knowledge aspects without any additional steps.

The technical documentation is much more complex, as it used to describe all aspects of the technology. Before information can be extracted a selection of text passages must be performed to reduce the amount of text to be analyzed.

Passages with relevant information are overview and introductory sections. Other passages, such as API documentations and implementation details can be skipped. However, technical documentation is written in different styles, so no overall matching selection process can be layed out here.

The selected passages are then analyzed with a text analysis method. The literature [9, 10] motivates two knowledge extraction concepts called *inductive category development* and *deductive category application*. The main difference is that the inductive approach is used to develop categories, where the deductive approach uses existing categories. Therefore the first approach is more suitable for the KMT, because categories (which are closely related to concepts for the knowledge base) have to be determined.

The inductive category development [10] is a step based approach. Step one is the definition of an initial research questions. For the KMT, the three knowledge aspects provide these research questions.

In step two, selection criteria for categories are defined on the basis of the three knowledge aspects, which are used as initial selection criteria. While reading the selected passages, a better understanding of the problem develops, which is used to refine the selection criteria [10].

The third step is for category development. To speed up this process, existing concepts from the knowledge base can be used as initial set of categories. In this step the selected passages are partially read. The parts to read have to be selected by the engineer [9–12].

The method [10] works in an incremental way. Therefore only 10% or less of the selected documentation is analyzed in a first run. Then (step four) a first revision of the categories is undertaken. If the categorization proves unsatisfactory, the steps two and three are repeated and more of the selected texts are examined until a satisfactory set of categories emerges.

In step five all selected text passages are paraphrased and categorized [13]. All collected phrases in the text analysis task are added to the informal knowledge description.

Alternatively to paraphrasing, a statistical identification technique for key sentences [14] can be used. Such algorithms are used in text classification and preprocessing for machine learning. However, these algorithms need keywords and training documents. Therefore, in a bootstrapping phase textual analysis is required.

3.2 Relating

The *relating* task is used in the style modeling procedure to find relationships of existing styles in the style repository. The knowledge modeling technique extends this step by finding relating properties, technologies, and design pattern. All these relationships are compiled as a list of relationships.

For example: The Spring Framework implements the Inversion of Control pattern. The sentence indicates a relationship between *the Spring Framework* and the *Inversion of Control* pattern. From the knowledge base the information can be retrieved that *Inversion of Control* is defined as an *ArchitectureFamily*. Therefore the relationship list contains the entry "*Spring* is a specialization of *Inversion of Control*".

Beside the relationships of variation and specialization, the knowledge base can also express exclusions or complements. Exclusion here means, that a certain platform or framework does not work together with another framework. And complement means, a technology or framework complements another framework to make a whole. Because the knowledge base is defined as an open system, other relationships can be introduced as well and therefore they should be added to the list of relationships.

3.3 Codification

The *codification* task is used to transform the *informal knowledge description* and the *list of relationship information* into a list of codified phrases which are collected in the *codified knowledge description*.

The sentences stored in the informal style description and the relationship information are analyzed to determine noun phrases. For example: Spring supports the Model-View-Controller pattern through Java Server Faces [15].

The terms *Spring*, *Model-View-Controller* pattern and *Java Server Faces* are noun phrases in terms of the English grammar. In the example sentence, there are different relationships. First, Spring *supports* Model-View-Controller. Second, the Model-View-Controller pattern *can be implemented with* Java Server Faces. And third, Spring *works with* Java Server Faces. To identify these relationships, the sentences have to be broken down in simple subject predicate object sentences. All subjects and objects found by this analysis task can be considered individuals or classes.

The next step is to find suitable classes for the determined individuals. Therefore existing classes have to be examined first. If no suitable classes exists in the knowledge base, new classes must be formulated.

Similar to the class finding step, the retrieved relationships have to be mapped preferably to existing properties, otherwise to newly created ones. Therefore the relationships have to be compared with existing property definition. In some cases relationships are expressed as the inverse of an existing property. In that case the relationship must be inverted and the existing property must be used.

Beside classes, individuals, and properties, rules can be extracted from the informal description. The rule modeling is more complicated, because rules cannot be determined on basis of syntactical elements. The phrases have to be checked on the pragmatic level. However, if a set of phrases conclude a rule in natural language, the linguistic objects in this rule can be mapped to individuals or classes, and linguistic predicates to properties from the knowledge base. Classes in SWRL are represented by unary predicates. The linguistic predicates are mapped on binary SWRL predicates, which are OWL object or datatype properties.

Conjunctions, like *and*, *but*, and *or*, must be translated into logical operators. The conjunction *and* is mapped to $\wedge$ (logical and) in a rule. The conjunction *or* results in alternative rules, where the *or* can be seen as separator. The conjunction *but* can be interpreted as a negation of all predicates in the phrase following the conjunction.

Beside the already explained linguistic elements, there are more constructs. However, a detailed analysis of natural language clauses and their transformation in horn clauses is not subject of this paper, but can be found in [16, 17].

All classes, properties, individuals, and rules determined in this task are compiled in the *codified knowledge description*.

3.4 Formalization

The *formalization* task is a straightforward process. All classes found in the *codified knowledge description* are formalized to OWL DL classes, all properties are formalized as OWL DL properties. At this point the engineer must be certain about the properties (functional, inverse, transitive) of these OWL DL properties. After formalizing the properties, the properties can be added to the newly formed classes. The last OWL DL formalization step is the formalization of the individuals.

To complete the formalization task, the codified rules are transformed into SWRL rules. In the codification task the sentences have been broken down into predicates and individuals. Conjunctions have been identified and their logic counterparts have been associated with them. All linguistic objects and predicates have been mapped to their corresponding OWL and SWRL constructs. Now the codified rules are transformed into horn clauses [18] and finally in SWRL and auxiliary OWL expressions.

Rules with negated predicates must be marked, because SWRL cannot handle negated predicates. For negated class predicates, a workaround can be applied using the ability of OWL to define complement classes. A complement class is the inverse of a class and therefore it represents the negation of a class predicate. However, negated properties cannot be modeled with OWL and therefore such rules must be skipped until RIF-BLD, a more powerful rule language currently developed by the w3c, becomes available and is supported by solver and reasoning programs.

3.5 Review

This final review task checks if the new concepts, individuals, properties, and rules are really appropriate for the described style. Also other styles might benefit from newly introduced concepts and properties. Therefore these styles have to be reviewed with these new concepts and properties in mind. This review process also minimizes possible fragmentation in the knowledge base.

4 Knowledge Querying Technique

The *knowledge querying technique* (KQT) is the counterpart to the KMT. While the KMT is used to extract information from various sources and store them in a knowledge base, the querying technique is used to get information from the knowledge base. As input source the target requirements are used. In some migration projects GQM models are used to evaluate styles. The questions from these models can also be a source for queries.

The KQT is divided in five steps (see Fig. 3). First, relevant information is extracted from the list of requirements and optionally from the questions formulated for the quality model. Second, the information is analyzed for linguistic objects and predicates. Third, based on these objects and predicates, horn clauses are formulated and augmented with SQWRL predicates to get useful results. Fourth, query clauses are send to the Jess [4] solver. And fifth, the result is checked using the query sentences from the first step. If the results are satisfactory to select a style, then the process ends. Otherwise the queries are refined and the codification, formalization, and querying steps have to be repeated.

4.1 Question Retrieval

The term *question* is, in this context, not limited to its linguistic definition, but sentences which represent queries or requirements, like *we need a Java-based style*, are also considered questions.

The engineer collects requirements and questions and reformulates them into question and query sentences, which are then processed by a paraphrasing technique [13] to remove redundant sentences. Paraphrasing is important, because requirements and questions may address the same issue and would therefore only bloat the later codification. The result of this step is a document containing only valid and unique query and question sentences.

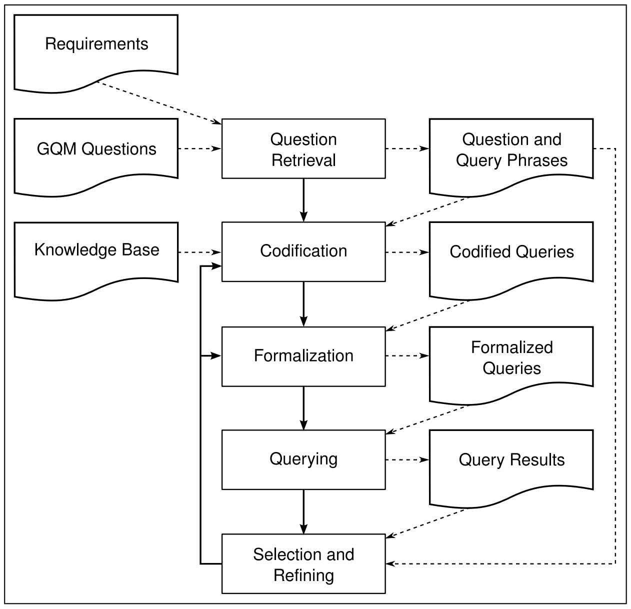


Fig. 3. The knowledge query technique, its tasks, and artifacts.

4.2 Codification

The query and questions sentences from the retrieval step are processed in the *codification* step to obtain structures which are close to horn clause like query rules. Therefore the engineer transforms the natural language sentences into predicates and individuals.

The linguistic objects are mapped to individuals or classes from the knowledge base. Classes are represented by unary predicates in SWRL (see KMT). And linguistic predicates are mapped to binary SWRL predicates, which are OWL object or datatype properties.

Similar to the KMT, conjuntions like *and*, *but*, and *or* must be translated into logical operators. However the resolving is done in the formalization task. Therefore the conjunctions are just marked.

4.3 Formalization

With the codified sentences from the previous step a set of horn clauses can be defined. Because horn clauses are disjunctions of literals, they can be combined to make a query more specific. Additionally SQWRL predicates can be added to further retrieve, sort, or manipulate the result.

First, each sentence is transformed into a horn clause with an empty consequent. Clauses based on alternative sentences (*or* conjunction) are still marked as alternatives, because this information is relevant for the last formalization step. Second, negations have to be resolved. And third, the clauses have to be combined. SWRL is not able to handle negation. Therefore the workaround from the KMT must also be applied here.

The last step is the combination of clauses. All clauses, which have to be satisfied together, are concatenated into one larger clause.

4.4 Querying

In the querying step the query clauses are send to the query interface of the knowledge base and the results are logged for later analysis.

4.5 Selection and Refinement

The results from the querying step are analyzed using the question sentences from the first step. If the results are sufficient one or more styles are selected as style candidates. If not, the queries are refined and the steps starting with codification are repeated.

5 Case Study

To evaluate and test the two introduced techniques, we used a case study [2] for the MidArch method [8] which developed architecture candidates for an existing regional trade information system *RegIS Online* [19].

We used the modeled architectural styles and collected meta-information for them with the KMT and we used the requirements mentioned in [2] as input for the KQT. As expected results the new method should return the similar conclusions as case study.

The styles modeled in [2] are *Inversion of Control*, *Sping*, *Avalon*, *Cocoon*, and *Cocoon with Flowscript*. First, we collected the informal style description from [2] and technical documentation [20–23]. Second, we selected sections and paragraphs from the documentation for further processing. Third, the selected passages were analyzed and processed according to the KMT. To illustrate the process we describe the analysis and knowledge modeling for a sub-set of the *Spring* style. A complete documentation can be found in [24].

5.1 Informal Knowledge Description

As input sources for the *Spring Framework*, we used the informal style description from [2], web documentation, and the technical documentation [21]. The enumeration below is an extract of the full list of phrases in [24]. We printed the objects (individuals) in italics to indicate the identification of individuals and categories (which is part of the relating and codification tasks).

1. The *Core package* provides the *Inversion of Control* and *Dependency Injection* features.

2. The *Context package* allows to access objects in a framework-style manner in a fashion somewhat reminiscent of a JNDI-registry
3. The *MVC package* provides a *MVC implementation* for *web-applications*
4. The *MVC package* allows the use of different *view technologies*, like *JSP*, *Velocity*, *Tiles*, *FreeMaker*, *XSLT*, *JasperReports* or *diverse document views*

5.2 Relating

The relating task revealed that *Spring* is a sub-class of *Inversion of Control*.

5.3 Codification

In the codification step, the phrases were transformed into codified expressions. The numbers in front of the expressions below indicate the relating phrases.

1–4 *Spring* hasPackage *Package* with values *SpringCorePackage*, *SpringContextPackage*, and *SpringMVCPackage*
1–4 *Package* (class) has restrictions: supportsPattern, supportsTechnology, providesTechnology
1 *SpringCorePackage* providesTechnology *Spring*
2 *SpringContextPackage* supportsTechnology *JNDI*
4 *SpringMVCPackage* supportsTechnology *JasperReports*, *JSP*, *Velocity*, *Tiles*, *FreeMaker*, and *XSLT*

The codification step revealed that different frameworks such as *Spring* and *Avalon* use different vocabulary to describe the same concepts and individuals. This resulted in definitions of classes and individuals as synonyms for existing ones. Also this step revealed that the quality of the technical documentation directly affects the quality of the result of the KMT.

5.4 Formalization

The formalization of the above codified expressions results in OWL DL source code in *abstract syntax notation* shown in Listing 1.1.

```
Class(Package partial owl:Thing
   restriction(supportsPattern someValueFrom(Pattern))
   restriction(supportsTechnology someValuesFrom(Technology))
   restriction(providesTechnology someValuesFrom(Technology)))

Individual(SpringMVCPackage type(Package)
   value(supportsTechnology JasperReports)
   value(supportsTechnology JSP)
   value(supportsTechnology Velocity)
   value(supportsTechnology Tiles)
   value(supportsTechnology FreeMaker)
   value(supportsTechnology XLST))
```

```
Individual(SpringCorePackage type(Package)
   value(providesTechnology Spring))

Individual(SpringContextPackage type(Package)
   value(supportsTechnology JNDI))

Individual(Spring
   type(intersectionOf(MiddleTierPlatform IntegrationPlatform))
   value(hasParent InversionOfControl)
   value(hasComponentType SpringBean)
   value(hasComponentType SpringApplicationContext)
   value(hasComponentType SpringBeanFactory)
   value(hasPackage SpringMVCPackage)
   value(hasPackage SpringCorePackage)
   value(hasPackage SpringContextPackage))
```

Listing 1.1. Formalization of the Spring architecture family based on the above extract.

One problem occurred in the formalization of the two Cocoon architectural families. Both families use the same technology, however they do not use the same set of component types from the technology. As middleware-oriented architectural families are coded as intersection of technology and architectural family, this resulted in a dilemma. To define the Cocoon technology correctly all component types must be added, however to define the two families correct. The basic Cocoon style does not have Flowscript as a supported technology. The derived style Cocoon with Flowscript has this property.

Another issue is the definition and separation of *Standard* and *Technology*. For our paper we used the following separation. A standard is the abstract definition of a technology. A technology is a concrete implementation of a standard or a specification. However, this distinction is not used in all documents available.

The KQT worked as expected with one exception. The requirements used vocabulary which was not present in the knowledge base. Therefore these terms were added and associated with the existing knowledge. This revealed that insufficient technology documentation (in this case the *Avalon* documentation) can lead to incomplete knowledge.

6 Conclusion and Future Work

The herein proposed techniques are designed for the MidArch method [8], which defines processes to handle migration and integration projects for middleware-based systems. However a detailed explanation of the method exceeds scope of this paper and can be found [25, 26, 8].

The evaluation of the proposed techniques revealed that the defined processes worked as intended. However, the idea of defining middleware-oriented architecture families as intersection of architecture family and technology introduced unnecessary limitations. This could be circumvented by defining middleware-oriented architectural families as sub-set of architectural families, which can handle sub-sets of technological properties in form of configurations. These configurations could be represented by association of the family to a technology and of component, connector, port and role types to components, connectors, ports, and roles of the technology.

Another imported result from the evaluation is, that the KQT should be extended by an optional knowledge retrieval step, which can be performed when the codified queries contain vocabulary which is not available in the knowledge base.

The two techniques are designed for the MidArch method, however the knowledge base could also be used in other contexts. As the knowledge base stores information on different technologies, it can be used during requirement engineering to search for standards and suitable these technologies. In a feature driven approach these features could be used to formulate queries and the knowledge base can show suitable technologies implementing the requested features.

In conjunction with projects, like [27], the knowledge base can be used to support developers and software engineers in the prototyping and implementation phase of a project. The knowledge base allows comparisons of technologies in respect to certain criteria, which is a common use case in the early stages of a software project where the software stack has to be determined.

Another possible future use can come from such knowledge bases, when best practice knowledge on requirements and goals are stored in a knowledge base as well. When such knowledge would be available in a set of OWL knowledge bases, these knowledge bases could be combined, and in combination they could point to solutions for a given software problem by presenting suitable technology combinations. The software engineer would then evaluate the results and modify the problem description until the presented solution can become the basis of the desired software architecture.

References

1. Nitto, E.D., Rosenblum, D.: Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In: ICSE '99: Proceedings of the 21st international conference on Software engineering, Los Alamitos, CA, USA, IEEE Computer Society Press (1999) 13–22
2. Giesecke, S., Bornhold, J.: Style-based Architectural Analysis for Migrating a Web-based Regional Trade Information System. In: CEUR Workshop Proceedings. Number 193 (2006) 15 — 23
3. Pahl, C., Giesecke, S., Hasselbring, W.: An Ontology-based Approach for Modelling Architectural Styles. Lecture Notes in Computer Science **Volume 4758/2007** (2007) 60–75
4. Jess Project, Sandia University: Jess, the Rule Engine for the Java Platform (Febuary 2008) http://herzberg.ca.sandia.gov/.
5. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language – Reference (February 2004)

6. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML (May 2004)
7. Shaw, M., Clements, P.: A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In: Proceedings of the 21st International Computer Software and Applications Conference, IEEE Computer Society (1997) 6 – 13
8. Giesecke, S.: Architectural Styles for Early Goal-driven Middleware Platform Selection. PhD thesis, Carl von Ossietzky University Oldenburg (2008) (submitted).
9. Gerbner, G., Holsti, O.R., Krippendorf, K., Paisley, W.J., Stone, P.J., eds.: The Analysis of Communication Content: Developments in Scientific Theories and Computer Techniques. John Wiley & Sons, New York (1969)
10. Mayring, P.: Qualitative Content Analysis. Forum Qualitative Social Research **Volume 1** (June 2000)
11. Krippendorff, K.: Content Analysis: An Introduction to Its Methodology. Sage Publications (1980)
12. Rosengren, K.E., ed.: Advances in Content Analysis. Volume 9. Sage Publications, Beverly Hills (1981)
13. Strobl, R., ed.: Wahre Geschichten? : zu Theorie und Praxis qualitativer Interviews. Nomos-Verlagsgesellschaft (1996)
14. Coenen, F., Leng, P., Sanderson, R., Wang, Y.J.: Statisitical Identification of Key Phrases for Text Classification. In Perner, P., ed.: Machine Learning and Data Mining in Pattern Recognition. 5th International Conference, MLDM 2007, Springer (July 2007) 838–853
15. Sun Microsystems: Java Server Faces Technology (2008) http://java.sun.com/javaee/javaserverfaces/.
16. Dale, R., ed.: Handbook of natural language processing. Dekker, New York (2000)
17. Jackson, P., Moulinier, I.: Natural language processing for online applications : text retrieval, extraction and categorization. Benjamins Publishing Company (2002)
18. Horn, A.: On sentences which are true of direct unions of algebras. Journal of Symbolic Logic **Volume 16** (March 1951) 14–21
19. REGIO GmbH: RegIS Online website (2008) http://www.regis-online.de.
20. Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern (January 2004) http://www.martinfowler.com/articles/injection.html.
21. Johnson, R., et al.: The Spring Framework - Reference Documentation. The Spring Project (2008) http://www.springframework.org/documentation.
22. The Apache Foundation: Excalibur Project, Avalon Framework (2007) http://excalibur.apache.org/.
23. The Apache Foundation: Apache Cocoon 2.2 (2008) http://cocoon.apache.org/.
24. Jung, R.: Ontology-based Metadata for MidArch-Styles. Master's thesis, Carl von Ossietzky Universität Oldenburg (2008) (submitted).
25. Giesecke, S.: Middleware-induced styles for enterprise application integration. In: Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR06), Los Alamitos, Calif., USA (2006) 334–340
26. Giesecke, S., Bornhold, J., Hasselbring, W.: Middleware-induced Architectural Style Modelling for Architecture Exploration. In: Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), January 2007, Mumbai, India, IEEE Computer Society Press (2007)
27. Institute of Information Sciences and Technology: The Web of Patterns Project (February 2008)