An Integrated Facet-based Library for Arbitrary Software Components

Matthias Schmidt[1], Jan Polowinski[1], Jendrik Johannes[1], and Miguel A. Fernández[2]

[1] Technische Universität Dresden, Nöthnitzer Str. 46, 01187 Dresden, Germany
{matthias.schmidt|jan.polowinski|jendrik.johannes}@tu-dresden.de
[2] Department of Broadband Service Platforms, Telefónica R&D, Valladolid, Spain
mafg@tid.es

Abstract. Reuse is an important means of reducing costs and effort during the development of complex software systems. A major challenge is to find suitable components in a large library with reasonable effort. This becomes even harder in today's development practice where a variety of artefacts such as models and documents play an equally important role as source code. Thus, different types of heterogeneous components exist and require consideration in a component search process. One flexible approach to structure (software component) libraries is *faceted classification*. Faceted classifications and in particular *faceted browsing* are nowadays widely used in online systems. This paper takes a fresh approach towards using faceted classification in heterogeneous software component libraries by transferring faceted browsing concepts from the web to software component libraries. It presents an architecture and implementation of such a library. This implementation is used to evaluate the applicability of facets in the context of an industry-driven case study.

1 Introduction

Reusing software components has always been central to software engineering. However, in practice, component reuse is still seldom implemented on a large scale. A reason for this is the lack of generic solutions that fulfill the needs of modern software development, where complex systems are not implemented code-centric and do not only rely on the reuse of source code and binary components. Instead, such systems are realised model-driven and models become equally important reusable components.

In the REUSEWARE[3] [1] project, we developed a generic solution to implement composition systems for model components defined in arbitrary modelling languages. Thus, we provided a technical solution that allows developers to treat all models[4] created during a development process of a complex system as components and store them in a library for reuse.

[3] http://reuseware.org

[4] In this paper we refer to all artefacts created in a model-driven process as *models*. This includes documents and source code.

However, a major problem in reuse in general, is locating a desired component in a huge component library. About two decades ago, Príeto-Diaz proposed [2, 3] the use of *faceted classification*, a concept from book libraries which was introduced by Ranganathan in the 1930s, for software component libraries. In a faceted classification, not an object as a whole, but different aspects (i.e. facets) of an object are described. For example, well-known facets used in book libraries are author, topic and publisher. Since Ranganathan, attempts were made to realise such classifications for specific component libraries (e.g. [4]), but there has been little interest in this area in the last decade.

Interestingly, concepts of faceted classification can be found today in many online systems, such as e-commerce systems like Ebay or Amazon. These systems make use of a faceted classification to enable *faceted browsing* that, in contrast to traditional web-search, allows for explorative browsing.

The data queried on the web is not very different from the models that make up complex modern software systems. In both cases, different languages and methods are used in combination to specify and compose data. Also, to integrate such heterogeneous data, standards (e.g., issued by the W3C or the OMG) support the creation of common base technologies and tools.

In this paper, we recapitulate Prieto-Díaz's idea of using faceted classification in software reuse by transferring faceted browsing concepts of today's web to software component libraries that meet the demands of model-driven development. For that we take a closer look at component and model libraries as well as facet technologies that are used today in Section 2. We present an architecture and implementation of a facet-based software component library that can handle heterogeneous models defined in arbitrary modelling languages in Section 3. The implementation is based on widely used technologies and standards: The Eclipse Platform [5], the Eclipse Modeling Framework (EMF) [6] and the OMG's MOF standard [7]. We also explore how the greater amount of structure in software components—compared to web data—can reduce the classification effort. To show that the facet-based library can be used to browse and search for different kinds of models, we evaluate it in the context of a case study from the telecommunications domain defined by Telefónica R&D in the European research project MODELPLEX[5] and discuss other applications of it in Section 4. Finally, we conclude in Section 5.

2 Foundations and Related Work

The idea of using a library to maintain a set of components is well known and popular in software engineering. However, component libraries for reuse (called *reuse libraries* in [8]) need to provide additional metadata about their content in order to help users in deciding which component or service does fulfil their needs the best.

As stated by Mili et. al. in their survey of reuse libraries [8], a good reuse solution requires to be efficient, accurate, user-friendly and general. Furthermore,

[5] `http://www.modelplex.org`

the survey classifies the use of facets as a descriptive method and characterises it using a number of criteria. In that way, it identifies the approach as a method of high precision, recall and flexibility and rates the difficulty of use as very low and the method's transparency to the user as very high. As we see these characteristics as crucial for a component library we argue to use faceted component libraries for reuse.

To emphasise that other implementation methods for reuse libraries have drawbacks we visit classic component libraries in Section 2.1, followed by a discussion of libraries for models in Section 2.2. Oriented at the mentioned requirements by Mili et. al., we aim to provide a facet-based library for model components. For that, we transfer well established faceted browsing techniques from online systems and present an implementation based on modelling standards and technologies. The library integrates seamlessly into the widely used software development and modelling environment Eclipse. Nevertheless, the solution is general by being independent of the language a model component is defined in. As a foundation of this solution, we introduce the main ideas of faceted classification in Section 2.3 and analyse early approaches as well as recent applications in Section 2.4.

2.1 Classic Component Libraries

As the main representatives of classic component libraries, we take a closer look at CORBA[6] and UDDI[7]. In principle, they implement different library approaches but have main features in common. They manage a database of components or services while users are able to register new components and search for existing ones. In order to search the database, these systems often implement a naming and/or directory service that give users the possibility to search by name (keyword) or id. As this requires detailed knowledge about the desired component or service there is a need for additional features to support users that do not have this information available. These users would not search by one concrete query but instead browse to get something adequate.

The CORBA middleware manages components as so called objects and offers a naming service to find them. Besides that, trading and property services allow a search based on component attributes [4]. Furthermore, a query service allows reading and manipulating queries on a set of objects using languages such as SQL or OQL. Although a search on the basis of attribute values is possible, CORBA does not provide a method to search on structured metadata due to a missing vocabulary of values. This does not allow for an efficient retrieval of components [9]. As a result, users require detailed knowledge about what they are searching for and might face a situation where effects such as synonyms, antonyms or plural forms complicate the search process.

The directory service UDDI acts as a library for web services in the domain of service oriented architectures (SOA). We see web services as a special case of

[6] `http://www.corba.org`
[7] `http://uddi.xml.org`

software components as they provide specific functionality over a well defined interface. To describe services, UDDI offers attribute values and enumerated classification [4]. This is implemented by so called White, Yellow and Green Pages which each focus on a specific service aspect. White Pages name attributes of the business that offers the service, Yellow Pages use standard taxonomies such as the North American Industry Classification System (NAICS) to classify business and service while Green Pages include technical details. Hence, to search a web service one can draw upon keywords, attribute values and enumerated classifications. As argued before, we do not see classifications by keywords or attribute values as methods of efficient component retrieval. However, with Yellow Pages UDDI also offers enumerated classification which provides a controlled vocabulary and eliminates effects such as antonyms or plural forms. Nevertheless, we do not think the taxonomies of the Yellow Pages to be adequate for classifying software components, since taxonomies are too large, inflexible and difficult to extend [9]. Although multiple taxonomies are allowed, the user is still required to classify his artefacts in an existing complex schema that might not be designed for his special purposes. Besides, UDDI itself does not support browsing a library and therefore it does not seem appropriate for an exploration by users who do not know in advance what is inside the library. The website seekda![8] adds concepts such as tagging and community evaluation to the UDDI's search engine. That also supports our claim that the UDDI's principles are not sufficient. Note that our work does not aim at providing automatic component selection and binding as UDDI provides for web services. For our work we are interested in the capabilities for manual searching performed by users.

2.2 Component Libraries for Models

In addition to classic component libraries, we shortly analyse the field of component libraries for models. In their *research roadmap* for model-driven development of complex systems [10], France and Rumpe mention the need for reuse of experience, libraries of model operations and full-featured model repositories. However, they do not explicitly request a method for intuitive browsing of model repositories (or libraries). To the best of our knowledge, there is no work aiming at building a *reuse* library (in the sense of Mili et. al.) for models in model-driven development. Surely, libraries specific to modelling languages and models (e.g., model libraries in SysML [11]) or specific to model operations (e.g., libraries of operation in Epsilon [12]) exist. But they are specific to a modelling language and do not integrate concepts for browsing and finding models. Since in model-driven development everything can be treated as a model (e.g., documents, model transformations, model management operations or metamodels) all reuse concerns identified in the mentioned research roadmap can profit from a reuse library for models that is independent of the language a model is defined in.

[8] `http://seekda.com`

2.3 Faceted Classification and Faceted Browsing

Faceted classification [13] combines principles from keyword classification and enumerated classification. Keywords (facet values) describing an entity are bundled into facets and each of these facets concerns only one single aspect to characterise the entity. Examples are shown in Figure 1 as well as Table 1 and 2. Some facets may be structured and form trees, each representing a single taxonomy, others may be flat [14, 15]. All facets as one create a multi-dimensional classification.

Faceted Browsing is a user interface paradigm based on flexible classification and has the following principles [16, 17]: The process of faceted browsing interactively constructs a query on the data while the user performs multiple simple refinement steps. At the beginning a complete set of items is presented, which is then reduced to a subset by making restrictions to the values of one or multiple facets (*zoom-in* navigation step). The subset can again be extended by taking back restrictions (*zoom-out* navigation step). Zoom-in and zoom-out navigation steps may be performed for all facets in any order. Note that this enables the user to choose his own navigation path—this is a main difference to fixed taxonomies which imply that the user follows the way the taxonomy was once constructed by its author. Another important feature of a faceted browser is the exclusion of empty result sets by construction. For this purpose only facet values that are available in the current result set are suggested as filtering options to the user.

As the system presents facets and facet values in the user interface, the user gets an impression of which options are on-hand. This way he learns about options he could not have named correctly in a textual query either because he has only partial knowledge of the domain or simply because the options did not come to his mind. Especially in this context, offering a description of the meaning of facets and facet values can further add to the guidance of the user.

2.4 Facets in Use

Many application examples show that facets define an intuitive classification schema. It is not only used in classic media libraries but was applied to software component libraries two decades ago. [4, 2] describe experiences made with component libraries that make use of faceted classifications. Although these examples characterise faceted classification as a promising approach, they can not directly be applied to todays' model-driven software development. At that time, software components were defined as programmatic functionality for reuse. Today, heterogeneous artefacts such as models, documents or binary components implemented in various languages need to be taken into account to support the whole model-driven development process. Furthermore, the Internet has changed the way components are delivered and today work is more and more shifted to a community rather than to single persons. These new aspects of component reuse require a new evaluation of faceted classification.

Since the first approaches for facet-based software component libraries were introduced, other fields of application made use of faceted classifications. Hence,

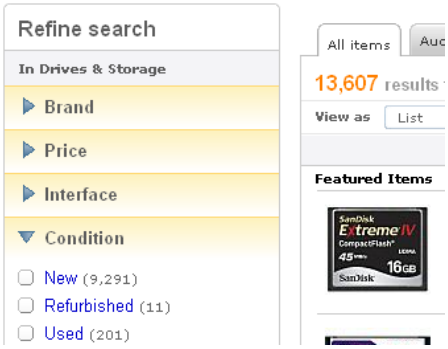

Fig. 1. Ebay uses facets to browse auctions

today there are many websites and desktop applications using this approach to browse huge amounts of data. Here, it is not always obvious that facets are used because the terms *category* and *filter* are often used as synonyms. This seems to be appropriate as they emphasise the structuring character and explain the way faceted browsing is performed.

[16] analyses a number of websites, web technologies and desktop programs that use faceted browsing. Applications such as iTunes[9] and foobar2000[10] or generic browsers such as Flamenco[11], Exhibit[12] and Longwell[13] show that the faceted browsing paradigm can be used in various fields of applications. In addition, websites such as Amazon[14], Google Base[15] or Ebay[16] use faceted browsing to give the user access to their dataset. Figure 1 shows Ebay as an example. Here auctions of flash memory drives are presented which can be browsed using facets such as *Brand*, *Price*, *Interface* or *Condition*. Depending on the auction's type other facets are shown which makes facets such as *Megapixel* or *Optical Zoom* available for Digital Cameras. To sum up, all these examples show that faceted browsing offers a flexible method of exploring arbitrary data.

3 A Facet-based Component Library

We see faceted classifications and faceted browsing, which has shown its applicability in various online systems, as efficient and user-friendly methods to search libraries of model components. To apply this in practice, a facet-based library

[9] http://www.apple.com/itunes
[10] http://foobar2000.audiohq.de/foo_facets
[11] http://flamenco.berkeley.edu/
[12] http://simile.mit.edu/wiki/Exhibit
[13] http://simile.mit.edu/wiki/Longwell
[14] http://www.amazon.com
[15] http://base.google.com/
[16] http://www.ebay.com

system is needed that is integrated into the user's software development and modelling environment.

This section introduces an architecture for such a system and a concrete implementation that is based on the Eclipse Modeling Framework (EMF) [6] and integrated into the Eclipse Platform. We chose these technologies over generic browsers mentioned in Section 2.4, because Eclipse provides a popular platform for software development and modelling. Thus, the library integrates seamlessly into the development and modelling environment, which turned out to be an important usability factor (cf. Section 4.1).

The architecture is oriented towards the principle of faceted classification and the user interface paradigm of faceted browsing discussed in Section 2.3. First, a domain expert—the *facet developer*—defines facets (Section 3.1) that can be used by *component developers* to classify components in a second step (Section 3.2). Third, *component users* browse the component repository by specifying faceted queries via zoom-in and zoom-out (Section 3.3). We captured the concepts of faceted definition, component classification and component browsing in a metamodel shown in Figure 2.

3.1 Facet Definition

We first discuss the concepts for *facet definition* that are shown in Figure 2 (b). The facet developer has to perform a domain analysis in order to specify terms and concepts of the domain in focus [18]. This leads to a number of `Facets` which are grouped in a `FacetDefinition`. A `Facet` consists of a name and a description that gives component developers an idea of the facet's semantics. Besides that, `Facets` own a set of `FacetValues` that have a name and description as well. These three concepts allow the facet developer to define and maintain facets and their vocabulary. `FacetDefinitions` are later available to component developers to create `ComponentClassifications` (cf. Section 3.2).

Based on the metamodel we defined graphical user interface tooling that can be used by the facet developer. The tooling is integrated into Eclipse and parts of it are directly generated from the metamodel using EMF. The tooling includes an editor that allows for creation of new facet definitions by instantiating the metamodel. It furthermore supports the facet developer in deleting specifications, removing, adding or editing facets as well as deploying specifications to component developers. Usually, facets have to be defined once for a specific component type (e.g., for one modelling language) to capture domain concepts of that component type. However, to capture the right domain concepts in `Facets` and `FacetValues`, experimenting with classifying concrete components is often required. Thus, having the facet definition tooling integrated in the same tool that is used for component development (which is Eclipse in our case) is helpful.

Standard Facet Catalog In contrast to domain-specific facets, there are facets to describe model components independently of their application domain. They are inspired by facet sets mentioned in [4, 3] and target syntax, semantics, composition interfaces and other implementation aspects of components. Table 1

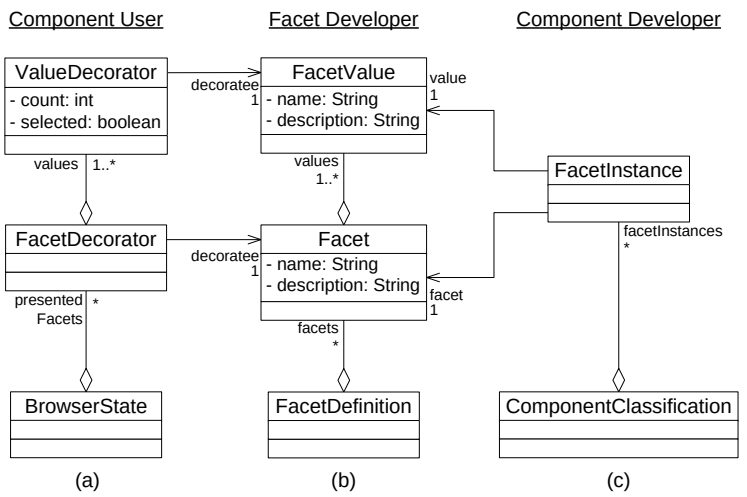

Fig. 2. Metamodel for faceted classification

shows an excerpt of these facets, which we provide as a standard catalog. They can be used by component developers directly for general faceted classification of software components. This standard catalog of facets is not closed or complete. It is rather expected that there are additional facets sufficiently adequate to classify model components independent of language and application domain.

3.2 Component Classification

Once facets are defined and deployed, components can be classified by component developers. A `ComponentClassification` (Figure 2 (c)) classifies one component and consists of a list of `FacetInstances`. A `FacetInstance` represents the usage of one facet to classify a component and encapsulates the `Facet` itself and one `FacetValue` that describes the component best. Note that only facet values that were assigned to the facet by the facet developer can be selected, thus ensuring that the vocabulary is controlled. This is inline with the definition of faceted classification in [4] which is not enforced by all faceted browsers.

For component developers, Eclipse-integrated graphical tooling is provided for component classification. Eclipse, which provides a wide range of editors to create and modify all kinds of models, acts as component development environment. Thus, a component developer can develop and classify model components in the same integrated environment. Figure 3 shows a typical component classification example. Here, a CIM model component (CIM is a domain-specific modelling language that we will introduce in Section 4.1) is created in the CIM editor integrated in Eclipse (a). Our tooling provides a special view (b) that is used to classify the component in the currently active editor. This view offers the opportunity to select available facets and choose one given value for each (c) to create a `ComponentClassification`. The available facets were specified

Facet	Description	Examples
Composition Role	The role the components plays when composed with other components	Port, Sender, Client
Information Hiding	The degree of encapsulation provided by a component	Whitebox, Greybox, Blackbox
Language	The language the component is modelled in	UML, SysML, AADL Java
License	The legal agreement the component is published under	GNU GPL, Mozilla Eclipse Public License
Maturity	The status of development or usability the component is in	Alpha, Beta, Released
System Layer	The system architecture's level where the component is to be used	GUI, Persistence, Core, Transport

Table 1. General facets to classify components

in a facet definition and loaded into the library beforehand. (In this example we use domain-specific facets that are described in Section 4.1.) Additionally, some attributes can be defined to add more information about the component (d).

Automated Classification The manual classification process in a facet-based library can be costly and error-prone. This is because component developers need to classify a potentially high number of components with facets defined by facet developers (which are potentially different persons). Errors in the usage of facets might occur if the semantics of a facet were not sufficiently defined. Besides that, manually created component classifications might become invalid when the component evolves.

These issues can be addressed by a rule-based automation of the classification process. This approach uses information retrieval to generate a faceted classification from the component itself. Software components qualify for this technique because they are very low on free text [2] and have an inner structure—in particular models that conform to a metamodel. This approach can relieve a component developer from classifying components. Furthermore, the facet developer that creates the facet, gains control over how it is used. That means no deep knowledge of the facet is needed by the component developer since the facet developer makes sure that the facet is used in the intended way. In the end, this approach allows to generate the classification at the latest point in time to ensure that it reflects the current state of the software component.

In our implementation, all components are represented as EMF models within Eclipse. The metamodels—that is, the languages in which the components are written—are all defined in Ecore (an implementation of the OMG's MOF stan-

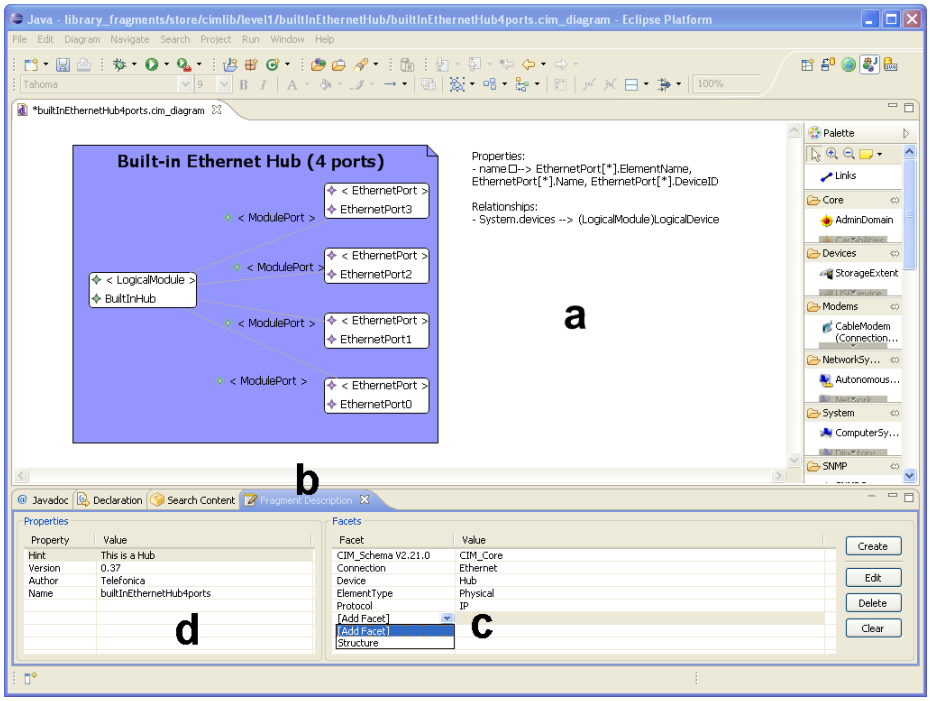

Fig. 3. The library's component classification

dard). Thus, all components can be inspected using the OMG's OCL [19] (which is aligned with MOF) as a query language. We allow facet developers to define automated classification rules in the form of OCL queries for arbitrary component types. Consequently, this approach is directly usable by facet developers who are familiar with the MOF and the OCL standards.

3.3 Component Browsing

After a faceted classification has been done and the components have been registered in the library, the component user can perform faceted browsing. The state of the browsing is captured in a `BrowserState` (Figure 2 (a)). A `BrowserState` holds a set of `FacetDecorators` where each refers to one of the facets that is currently explored by the component user. `ValueDecorators` represent the `FacetValues` the component user specifies to narrow down his search. Furthermore, `ValueDecorator` consists of a counter that indicates how many components will remain in the result if the user selects the `FacetValue` and a flag to represent the selection.

Following Figure 2 (a), we implemented a faceted browser that provides different facilities for the zoom-based exploration process (cf. Section 2.3) and supports different ways to present browsing results, following the works [16, 17]. The

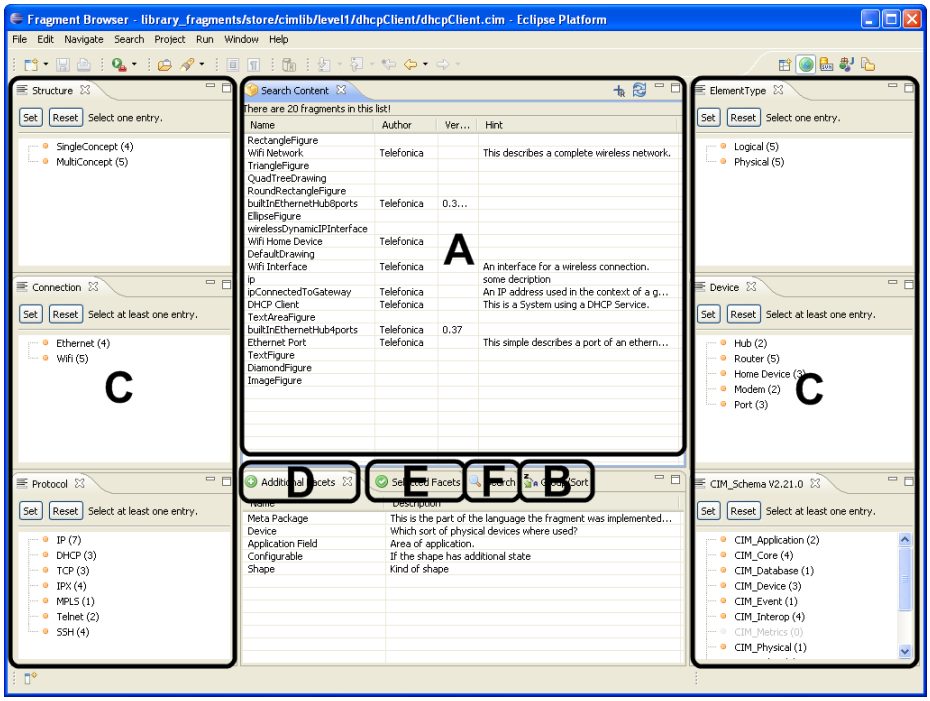

Fig. 4. The library's faceted component browser

features range from special widgets to present facets, over a free-text search to features such as grouping and sorting. Figure 4 shows our faceted component browser with important parts marked. These parts include the main functionality of a faceted browser, which are the result view (A), a grouping and sorting facility for the result view's entries (B) as well as six widgets to present available facets and their values (C). As there might be more than six facets available, a separate view lists the others not presented (D). While the user selects facets and values to perform zoom-in and zoom-out steps with (C) and (D), the current search query is shown in another view (E). Finally, a search view gives the opportunity to perform a free-text search over available facets and classifications (F) [15]. These features define a faceted browser that can be used to search arbitrary model components classified in the way shown in Section 3.2.

After the component user has found a suitable component, he can directly open other views or editors to inspect or to reuse the component. In our evaluation, we used the browser together with Reuseware, which is also integrated into Eclipse and provides a graphical composition editor. The component user can directly drag&drop components from the result view (A) into Reuseware's composition editor. Thus, the browser integrates tightly with the component users development environment.

4 Evaluation

To show the applicability of our facet-based library, we tested it with different models defined in different modelling languages. In Section 4.1, we describe an evaluation we performed with a domain-specific modelling language, where we collected feedback from the domain experts on the usability of our approach and our implementation. Furthermore, in Section 4.2, we discuss other types of models and artefacts that can be browsed with our approach.

4.1 Evaluation with Telecommunication Domain Experts

We performed an evaluation of our facet-based library in the context of a case study defined by Telefónica R&D in the European project MODELPLEX [20]. In the case study, Telefónica uses an EMF implementation of the Common Information Model (CIM) [21] and an Eclipse-integrated graphical editor to define graphical models of telecommunication networks [22]. In earlier work [23], we created a composition environment with REUSEWARE that is also integrated into Eclipse. This environment can be used to define reusable CIM model components and compose them to larger network models. In this evaluation, we provided the domain experts at Telefónica with the library tooling presented in this paper. To gain a first feedback, their task was to classify 20 CIM model components and then browse for components using the faceted browser.

In preparation for the evaluation, we created a set of domain-specific facets for the CIM language. For that we performed, in collaboration with the domain experts, a domain analysis and found a set of six facets (cf. Table 2). In addition to the general facets (cf. Table 1), they allow a classification specific for the telecommunication domain. A telecommunication expert can now use our integrated tooling to classify CIM models using the domain facets. Thus, he works in the terminology of his domain and does not need any knowledge about, for example, source code components or their classification. Other telecommunication experts can then use these domain-specific facets with our faceted browser to browse a library of CIM model components.

The remainder of this section consists of three parts, where we summarise the feedback we got from interviewing the domain experts concerning domain facet definition, component classification and component browsing respectively.

Domain Facet Definition The experts recognised facets as a useful approach for classifying CIM components in general. However, they see a potential weakness of the approach in the fact that facet developers bear a huge responsibility. First, these developers restrict facets and facet values that are the base for all later classifications and browsings. Second, they need to clarify the meaning of facets and values and should take the component developers' perspective into account. These aspects indicate that it is crucial for both, classification and browsing to have well defined facets available.

Facet	Description	Examples
CIM-Schema	Uses CIM specific terms to classify the component	CIM-Core, CIM-User CIM-Interop
Connection	Names the main connection used by the component	Ethernet, Wifi, Bluetooth
Device	Describes which sort of device is used by the component	Hub, Router, Modem
Element Type	Distinguishes between conceptual and real life components	Logical, Physical
Protocol	Names the main protocol used by the component	IP, DHCP, IPX, SSH Telnet
Structure	Gives a hint about the component's inner structure	SingleConcept, MultiConcept

Table 2. Telecommunication specific facets

Component Classification Faceted classification with its restrictive character[17] appears to be an adequate method for structuring a component library for the domain experts. If the facets are well defined they can support even a large number of component developers to classify their work without creating anomalies such as synonyms, antonyms or plural-forms. In addition to that, the experts pointed out, that providing domain knowledge as facets and facet values can simplify work especially in a huge domain such as telecommunication. This is because component developers and users do not have to remember all domain concepts on their own.

The automated classification appears to be a very useful approach for practical use. Rather than classifying huge sets of components by hand, the domain experts, in the roles of component developers, want to use as much automation as possible. Therefore, rules must be specified that cover important aspects of the domain. We identified many opportunities for CIM model components to specify such rules for automation (e.g., for the facets CIM-Schema or Structure).

Together with the domain experts, we identified one particular application of automation rules as an interesting alternative to using the specific classification tool. In the case of CIM components, adding notes to a graphical component diagram was a common method used by the domain experts. These notes contained information that could be extracted and translated into facet values. This was seen as a useful feature by the domain experts since it gives them the opportunity to define facet values directly in their models. This supports our argument that a tight integration of development environment and library system is crucial. All in all, the automated classification support was seen as a critical feature for broad industrial acceptance of faceted classification by the domain experts.

[17] Restrictive with respect to the controlled vocabulary.

Component Browsing Faceted browsing was received by the domain experts as an intuitive and user-friendly method to search in a huge repository of components. They acknowledged that step-wise searching in a faceted browser supports component users that think in the problem space rather than in the solution space. As transferring ideas between both worlds is a major challenge in finding the right component for reuse, presenting facets and values can help. The domain experts, who used the composition environment for CIM without the facet-based library beforehand, stressed the importance of integrating the library system into the composition environment. For them it was very important that a discovered component was directly reusable from the search result view of the component browser.

Nevertheless, the experts missed some features while testing the browser. The browser always constructs a query using logical AND concatenation of all selected facet values. The domain experts encountered cases, where there was a need to express that a facet value should NOT be set or where OR concatenations would be desirable. They suggested that the browser could be improved in the way that the component user selects a facet value that should not be met by the desired components or other configuration facilities, in order to influence the construction of the actual queries based on the selected facet values. Ultimately, the domain experts also suggested that for complex searches a SQL-like query language over the facet data would be helpful for experienced users. Nevertheless, the faceted browsing process has shown to be intuitive as it supports the user in various ways.

The overall results of the evaluation are positive. In particular, the following points were stressed:

- Faceted classification and browsing are promising methods to structure and explore libraries of domain-specific model components.
- Automatic rule-based classification appears to be important for usability and acceptance of the library system.
- The integration of the library system with component development and composition environment is important to support the reuse process.

4.2 Evaluation Using Other Model Component Types

The previous section showed the applicability of our approach to one kind of domain-specific model components. To support our claim that faceted classification and browsing can be used for arbitrary types of model components and that this is supported by our implementation, we tested our approach and implementation with different models, documents and code defined in different languages.

We experimented with languages and components used in the demonstrator system we realised in [24]. There, we performed a component-based and model-driven development of a system using different kind of components including OpenOffice documents, UML models, models defined in graphical and textual domain-specific languages and Java source code.

For all these component types, EMF metamodels and Eclipse-integrated tooling exists. As our implementation was created on the same platform we were able to classify components in their development environment. One interesting point to mention is, that we were able to define facets that were specific to the development process but not to a specific modelling or implementation language. For instance, it was possible to relate each component to one use case in the system. Thus, we defined a facet *UseCase* and classification rules that identified which component was related to which use case. We were then able to use the browser to identify all components related to a specific use case.

5 Conclusion

This paper presented a new approach to facet-based software reuse libraries that takes the requirements of model-driven software development practice into account. The novelties of our approach, compared to earlier facet-based library approaches, are the integration of modern faceted browsing concepts from online systems and the support for software components of arbitrary languages, which is in particular important for model-driven development where models defined in different languages are the components.

We presented an implementation that is integrated into the widely used Eclipse development and modelling environment. Since a variety of languages and tooling for Eclipse and the EMF does already exist, many developers can directly use our implementation in an integrated manner without adaptation effort. This was also vital to transfer our research results into practice.

Our evaluation with the Telefónica domain experts showed that the approach is applicable in practice to browse libraries of domain-specific model components. The results stress the importance of having the library system tightly integrated into the development and composition environment. This improves the usability, since the users have all tools needed available in a single environment. In our case, these tools are Eclipse editors and the Reuseware composition tooling.

The first evaluation and experiments we performed can only indicate the potential of an integrated, generic facet-based software component library system. Thus, in the future, we plan to optimize our implementation with regards to performance and to conduct further evaluations on larger component collections.

6 Acknowledgments

This research has been co-funded by the European Commission in the 6th Framework Programme project Modelplex contract no. 034081 (cf. `www.modelplex.org`) and the European Social Fond / Free State of Saxony, contract no. 80937064.

References

1. Heidenreich, F., Henriksson, J., Johannes, J., Zschaler, S.: On Language-Independent Model Modularisation. In: Transactions on Aspect-Oriented Development VI. Volume 5560 of LNCS., Springer (October 2009) 39–82

2. Prieto-Díaz, R.: Implementing faceted classification for software reuse. In: Communications of the ACM. Volume 34(5)., ACM (1991) 88–97
3. Prieto-Díaz, R., Freeman, P.: Classifying Software for Reusability. In: IEEE Software. Volume 4(1)., IEEE (January 1987) 6–16
4. Poulin, J.S., Yglesias, K.P.: Experiences with a Faceted Classification Scheme in a Large Reusable Software Library (RSL). In: Proc. of COMPSAC'93, IEEE (November 1993) 90–99
5. Eclipse Foundation: Eclipse platform technical overview (April 2006)
6. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework, 2nd Edition. Pearson Education (January 2009)
7. Object Management Group: MOF 2.0 Core Specification. `http://www.omg.org/spec/MOF/2.0` (January 2006)
8. Mili, A., Mili, R., Mittermeir, R.T.: A survey of software reuse libraries. In: Annals of Software Engineering. Volume 5., Springer (January 1998) 349–414
9. Rao, C.G., Niranjan, P.: An integrated classification scheme for efficient retrieval of components. In: Journal of Computer Science. Volume 4(10)., Science Publications (2008) 821–825
10. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: FOSE'07: 2007 Future of Software Engineering, IEEE Computer Society (May 2007) 37–54
11. Object Management Group: SysML 1.0 Specification. `http://www.omgsysml.org` (September 2007)
12. Kolovos, D.S., Paige, R.F., Polack, F.: The Epsilon Object Language. In: Proc. of ECMDA-FA'06. Volume 4066 of LNCS., Springer (July 2006) 128–142
13. Priss, U.: Faceted Knowledge Representation. In: Electronic Transactions on Artificial Intelligence. Volume 4. (2000) 21–33
14. Allen, R.B.: Retrieval from facet spaces. In: Electronic Publishing. Volume 8(2&3). (June 1995) 247–257
15. Hearst, M.: Design Recommendations for Hierarchical Faceted Search Interfaces. In: ACM SIGIR Workshop on Faceted Search. (August 2006)
16. Polowinski, J.: Widgets for Faceted Browsing. In: Proc. of HCI'09. Volume 5617 of LNCS., Springer (July 2009) 601–610
17. Sacco, G.M., Tzitzikas, Y.: Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience. Springer (August 2009)
18. Prieto-Díaz, R.: A Faceted Approach to Building Ontologies. In: Proc. of IRI'03, IEEE (October 2003) 458–465
19. Object Management Group: Object Constraint Language 2.0. `http://www.omg.org/spec/OCL/2.0` (May 2006)
20. MODELPLEX Project: Deliverable D1.1.a (v3): Case Study Scenario Definitions. `http://www.modelplex.org` (March 2008)
21. Distributed Management Task Force Inc. (DMTF): Common Information Model Standards. `http://www.dmtf.org/standards/cim` (January 2010)
22. Evans, A., Fernández, M.A., Mohagheghi, P.: Experiences of Developing a Network Modeling Tool Using the Eclipse Environment. In: Proc. of ECMDA-FA'09. Volume 5562 of LNCS., Springer (June 2009) 301–312
23. Johannes, J., Fernández, M.A.: Adding Abstraction and Reuse to a Network Modelling Tool using the Reuseware Composition Framework. In: Proc. of ECMFA'10. LNCS, Springer (June 2010)
24. Johannes, J.: Controlling Model-Driven Software Development through Composition Systems. In: Proc. of NW-MODE'09. (August 2009)