Modeling and Utilizing Quality Properties in the Development of Composite Web Mashups

Andreas Rümpel, Vincent Tietz, Anika Wagner, and Klaus Meißner

Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{andreas.ruempel,vincent.tietz,anika.wagner,
klaus.meissner}@tu-dresden.de

Abstract Ubiquitous Web resources and their manifold combinability are causing app development with current Web mashup platforms to be a challenging task, especially for low-skilled Web users. Hence, mashup composition demands for expressing quality requirements to facilitate selection and customization processes. Existing quality metamodels only provide guidelines or abstract categories, but neglect mashup-specific measuring directives and machine-readable representations. Therefore, we present a tailored quality property metamodel for composite Web mashups. We show, how it supports different settings of mashup development and execution. Finally, we demonstrate the metamodel's deployment in a mashup infrastructure and its utilization in different use cases.

1 Introduction

Modern mashup platforms, providing easy-to-couple mashup components, invite Web users with no programming skills to build their own applications with few time effort. Unlike in traditional software engineering, the mashup paradigm does not involve distinguished test departments with considerably high human and time resource costs to ensure quality requirements. Instead, mashup applications are created very situationally, since the paradigm strives for rapid development. Thus, to support instantly available and usable mashups, enforcement and monitoring of quality requirements has to be ensured by the development and execution platform. At least, beyond basic model-based application composition, cf. [1], quality assurance on an interface level of mashup components can be handled by employing semantic interface descriptions. Yet, only basic data type compatibility can be verified during application development.

Quality requirements may cover a variety of scenarios, such as user-driven filtering of candidate mashup components, device-specific requirements or resources availability at runtime. As a foundation, they have quality properties in common, which apply to mashup components or applications in question. Thus, a mashup-tailored *quality property metamodel* has to specify eligible properties, which can be provided or assigned by measuring. Without loss of generality, Figure 1 illustrates three example use cases for different kinds of quality requirements specification in a mashup development platform.

Q.Z. Sheng and J. Kjeldskov (Eds.): ICWE 2013 Workshops, LNCS 8295, pp. 54–65, 2013.

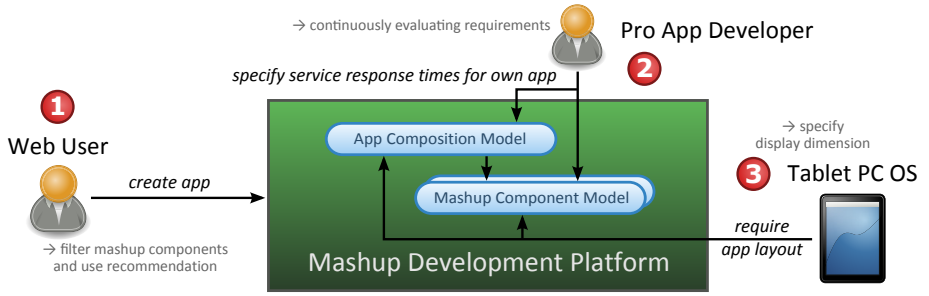

Fig. 1. Quality requirements use cases with unified mashup platform and models

In use case ①, a Web user without advanced skills or experience in developing applications with programming languages wants to find appropriate mashup components for a certain task. He uses an EUD (end user development) platform such as [2], which provides a browser for available mashup components. Assuming a huge number of mashup components, the user desires to get a recommendation involving custom quality requirements, pointing at component quality properties with the help of frontend. Use case ② is initiated by a professional app author, being familiar with modeling mashups. He wants to specify application-wide and component-specific quality requirements, which should hold at runtime. Therefore, a requirements evaluation module triggers context monitors to observe different measurable quality properties like response times of service components. Quality property values will have to be evaluated continuously at runtime. Since quality requirements are not limited to originate from human stakeholders, use case ③ involves a mobile device as a requirements source. On the application level, the device requires a certain application layout and different display size parameters. As a consequence, the mashup has to arrange its mashup components, which themselves should adapt their internal presentation, at least at deployment time. Additionally, the mashup platform could exchange components with regard to those device-specific requirements.

To support all outlined use cases by one holistic approach, we propose a quality property metamodel as a basis for customized requirements specification and evaluation in composite Web mashups. Our model, see Section 3, is able to describe quality properties for different quality requirements stakeholders, i. e., mashup components and mashup applications and it can be easily extended to consider the runtime platform, the device which the application is running on and the user profile. Therefore, we introduce mashup-specific quality properties completing non-functional properties known from Web services models, cf. related work in Section 2. Hence, quality properties may be used during development, at deployment time and at runtime. Section 4 shows their deployment in a requirements-aware Web mashup infrastructure. The presented use cases are serving as example scenarios to demonstrate model instantiation and management of its contents within an established platform for composite Web mashups, see Section 5. Finally, Section 6 concludes this paper.

2 Related Work

According to [3], quality requirements engineering effort and tooling support decreases when regarding application paradigms in the following order: traditional software systems, Web applications and Web mashups. Thus, there is a huge overall demand on solutions providing requirements specification and evaluation support particularly for the mashup paradigm of application engineering. In general, *quality properties*, ideally accompanied by a defining *metamodel*, might be assigned to each artifact or stakeholder of a software system. In traditional software engineering, a number of quality models have been proposed by the Consortium for IT Software Quality (CISQ), ISO and IEC, e. g., ISO/IEC 9126-1 and the subsequent ISO/IEC 25010 [4]. They are complemented by academic approaches, such as from McCall [5] and Grady [6]. Some properties provide *metrics* such as *customizability* from ISO 9126, which is calculated by the division of the number of functions which can be customized during operation by the number of functions that require the customization capability. However, for the majority of properties, implementation is hard to accomplish in a real system, since they are lacking metrics and scales for automated processing. Further, they are very general and need to be tailored for each specific class of software.

Especially in Web services description, cf. WSMO [7], and matching [8], models for non-functional properties have been proposed several years ago. Although their models are quite extensive, comprehensive implementations are missing. Since they describe Web services, crucial application-specific properties like UI concerns are neglected. Thus, they are not suitable for mashup applications out of the box. An example for a specific quality model for Web applications is provided by [9] and [10]. Based on ISO/IEC 9126, it comprises new properties such as information quality, loyalty and actuality. Since they focus on Web-based applications in general, quality properties are not discussed at a level of granularity typical for mashup components. [11] focuses on *new generation WebApps* and standard completeness of their quality model. However, customized requirements suitability of the described properties is only discussed to little extent. A dedicated quality model for mashup components has been introduced by [12], which focuses on interface quality, data quality and presentation quality. An extension for whole applications was proposed in [13]. Following the previously mentioned ISO standard, provision of quality properties was narrowed down to scaling values within integer values depending on the presence of particular features such as API keys, SSL support or user accounts.

Although quality property modeling approaches, especially for Web services, contain many useful elements, they are often limited to be used as a supporting feature to facilitate service matching. Moreover, they do not contain UI-specific properties, which are essential for mashup applications. First dedicated mashup quality property models compensate that issue to a limited extent. On the downside, their properties are recruited from ISO 9126 and lack concrete scales and measuring facilities to integrate them in a quality-aware mashup platform being helpful in the presented use cases. Thus, a quality property modeling approach with special regard to support customized quality requirements is still missing.

3 Web Mashup Quality Property Metamodel

Since existing quality models are either less formal or lack mashup-suitable properties, we now introduce a tailored quality property metamodel[1] for composite Web mashups. In this regard, we assume the following design principles and requirements. According to the application paradigm of presentation-oriented Web mashups, we include end-users, who are able to conceive and to specify quality requirements, into our target group. In addition, mashup component developers may need to utilize the quality model to describe their components accordingly. Therefore, its characteristics respectively its serialization and presentation need to be understandable by both mashup composers and component developers. With use case ③ from Section 1 in mind, fully automated processing without any human interaction should be possible as well. Furthermore, machine-readable representation is crucial to enable context monitoring and adaptation, e. g., by exchanging composition fragments triggered and invoked by the runtime environment. As mentioned previously, the quality model also needs to consider certain elements and concerns, that are missing in prevalent approaches.

To design the metamodel, we reviewed properties described in existing quality models, see Section 2, regarding the characteristics of mashup applications and the suitability of the metrics to be interpreted by a runtime environment and component recommendation facilities. Thereby, we disregard those with less relevance for the mashup application type. For example, *analysability* [4] cannot be provided by black-box mashup components. Following mashup component models, each component should be easy to install. Therefore, we consider *portability* [5] also as less relevant. However, we also adopted existing properties such as *interoperability*, *learnability* and *timeliness* from [12] and *actuality*, *security* from [10]. We added *access mode*, *data storage*, *style*, *bandwidth*, *source* and *data traffic* to address UI and Web characteristics appropriately. Their representation in our model is described in more detail in the following section as well as the metamodel's structure and noteworthy modeling features.

3.1 Structure of the Quality Metamodel

Quality properties may be carried by different stakeholders in a Web mashup infrastructure. We now focus on the mashup application as well as its constituent components as property carrying entities. Therefore, we treat them in a modularized fashion, as illustrated in the metamodel overview in Fig. 2. Therein, we define the following three model parts, i. e. ontologies:

Basic Property Ontology. defines the concept of a property carrying entity from which all other properties are derived. It comprises about 50 properties, which can be used for describing components and applications, e. g., *actuality*, *response time*, *usability*, *learnability* and *traffic*.

[1] Metamodel OWL files: `http://mmt.inf.tu-dresden.de/models/qpm.xhtml`

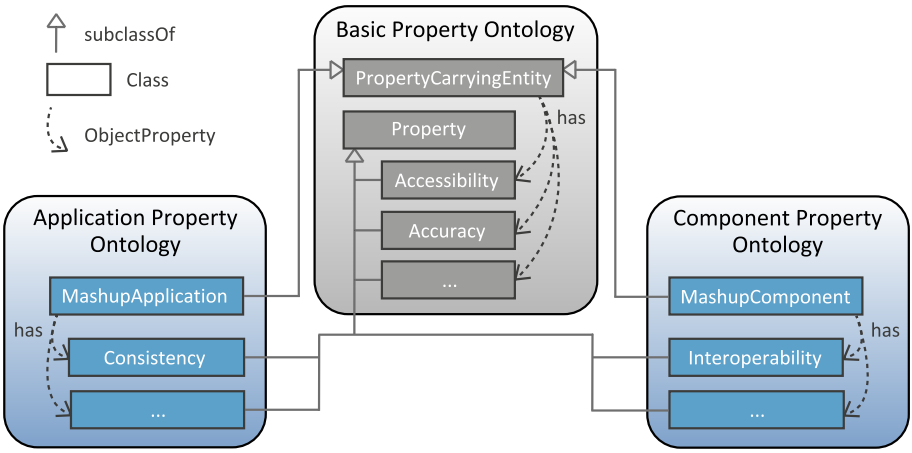

Fig. 2. Quality property metamodel structure overview

Application Property Ontology. defines properties of the mashup application such as *consistency* and *component appropriateness*. In addition, this sub-model contains three properties, that are special for applications: *component suitability*, *consistency* and *component number*.

Component Property Ontology. defines properties of mashup components. Additionally, it contains seven attributes, that are special for components, e. g., *access mode*, *interoperability* and *isUI*.

3.2 Metamodel Elements Nature and Features

Our metamodel provides relations between its properties such as the sub-class relationship, e. g., *interoperability* incorporates *data exchange formats*, *programming language* and *protocols*. Further, each property is associated with formal metrics and information on how the data is acquired. We distinguish provided, collectible and measurable properties. Provided properties are statically defined either by the application composer or the component developer, such as the supported types of *security* mechanisms like *authentication*. Collectible properties are accumulated over time, e. g., with the help of user ratings or by the runtime environment. Values are calculated by a function such as average, minimum or maximum, e. g., *usability* by $\varnothing$(UserRatings). Measurable properties can be acquired instantly and fully automated using context monitors. For certain properties, *concerns* such as API, UI data, backend data, functionality, service or UI are eligible to be assigned. *API* represents the interface providing access to the functionality of a specific mashup building part. Component-internal application logic is represented by *backend data*. In contrast, *UI data* refers to visible data on the user interface of the component or the application, while *UI* targets the presentation of UI elements itself. Of course, those concerns can only be assigned if a UI mashup component is in consideration. *Functionality* represents the general

competences, the component or application can execute. Properties, which refer to third-party Web service calls of mashup components, are assigned to *service*. With the help of those concern assignments, very similar quality properties can be semantically separated at a conceptual and implementation level.

Finally, the metamodel provides a descriptive text definition, a data type, a default value as well as value ranges and a unit for each property. As an example, *data traffic*, which is accrued over time and assessed by the metric $(\sum_{i=1}^{N} \text{DataTrafficPerSecond}_i) \div N$ (N being the whole usage time in seconds) has the unit KB/s and a default value of infinite, since this is the worst value. Additionally, we defined linguistic variables like *low*, *middle* and *high*, that are assigned to concrete values using a membership function, to facilitate configuration of fuzzy specification of quality requirements for our properties.

3.3 Metamodel Content

Table 1 outlines the metamodel's subset of concern-aware properties. We distinguish properties collected from existing quality models, properties adopted to the mashup domain and newly added ones. The semantics of *actuality* was e. g. extended to track the frequency of refreshes from a background service. The added property *source* provides means to describe the origin, i. e., an organization or URI of the used Web services. Usually, this information is not explicitly available for users, whereas it is difficult to put trust in the described artifact. In this context, *security* is also an important aspect, that is explicitly considered in our model by the six sub-properties *authentication*, *authorization*, *data storage*, *data encryption*, *traceability* and *confidentiality*. The security property itself indicates with an integer how secure a component or application is. It can be measured as |usedSecurityMechanism|. To provide means for adaptation, we introduce *bandwidth* and *data traffic* that both belong to the concern of backend data. The former describes the data transfer rate, which the component or application needs to run optimally, whereupon the application bandwidth is assessed by $max(\text{Bandwidth}_{\text{Components}})$. The latter describes how much data traffic the component or application consumes while running. This also can be utilized to avoid data traffic limits, being of relevance especially on mobile devices.

In existing quality models, the meaning of *consistency* refers to the UI or the backend data. However, in mashups this distinction is not relevant for the user. This is why we decided to consider both equally under this property. Further, we extend *completeness* to the API, UI and background data. For each concern, user ratings can be collected, while the average of this ratings forms the property value. Also, the *availablity* is extended to backend services. The metric for this concern for components is $(1 - \text{ConnectionErrors} \div |\text{AccessesToService}|) \cdot 100$. *Robustness* does not only handle exceptions after user input, but all exceptions. It can be calculated as follows: $(1 - |\text{Exceptions}| \div \text{TotalUseTime}) \cdot 100$. Finally, *timeliness* describes, if tasks are fulfilled within a certain time, that is the *deadline* defined by users or developers. For components it is assigned by $|\text{ResponsesInTime}| \div |\text{Measurements}| \cdot 100$ and for applications by $\varnothing(\text{Timeliness}_{\text{Components}})$. Beside these concern-related properties, we also

Table 1. Quality properties with special concerns

	API	UI data	Backend data	Functionality	Service	UI	App./Comp.
Provided							
▲ Interoperability [12]	●		●				▪
▲ Data exchange formats [12]			●				▪
▲ Programming language [12]	●						▪
▲ Protocols [12]	●						▪
◆ Actuality (Currency) [10]		●	○				▦▪
▼ Access mode			●				▪
▲ Credibility [10,9]		●	○		○		▦▪
◆ Security [14,4,15,5,10,9,16,17]			●	●			▦▪
▲ Authentication [17,14]				●			▦▪
▲ Authorization [17,14]				●			▦▪
▼ Data storage			●				▦▪
▲ Data encryption [17,14]			●				▦▪
▲ Traceability [17,14]				●			▦▪
▲ Confidentiality [17,14]				●			▦▪
▼ Style						●	▦▪
▼ Bandwidth			●				▦▪
▲ Familiarity [9]						●	▦▪
▼ Source					●		▦▪
Collectible							
▲ Component suitability [13]	●	●	●	●	●	●	▦
◆ Consistency [5,9,6,13]		●		●		●	▦
▲ Adaptability [4,15,5,9,6]		●		●		●	▦▪
▲ Usability [9,5,6,4,15,13,12]	◗					●	▦▪
▲ Learnability [9,4,15,12]	◗					●	▦▪
▲ Accuracy [10,9]		●					▦▪
◆ Completeness [17,5,10,4]	◗	●	●	●		●	▦▪
▲ Satisfaction [9]	◗	●	●	●	●	●	▦▪
▲ Accessibility [10,9]		○				●	▦▪
Measurable							
▼ Data traffic			●				▦▪
◆ Reliability [4,15,5,9,16,17,14,6]		○	○	●	●	○	▦▪
◆ Availability [4,15,9,16,17,14,10]				●			▦▪
◆ Robustness [17,14]		○	○	●	●	○	▦▪
▲ Response time [4,15,9,16,17,14,6]				○	●	●	▦▪
▲ Stability [15]				●			▦▪
◆ Timeliness [12,13]				○	●	●	▦▪
▼ Deadline				○	●	●	▪

Legend

▲ from pre-existing models
◆ meaning slightly changed
▼ new property
▦ relevant for applications

● concern is affected directly
○ concern is affected indirectly
◗ concern is affected only for components
▪ relevant for components

incorporated metadata of components and applications, which can be treated in the same way during model evaluation. Examples are *name*, *author*, *certificate*, *development date* and *keywords*, whose values are provided by the author.

4 Requirements-Based Quality Evaluation

Many regarded metamodels for quality properties lack either implementation and real-world use cases or they are not field-tested to work within an application development or runtime infrastructure. Thus, this section shows the metamodel's integration into a quality-aware platform for composite Web mashups and therefore demonstrates its suitability in a variety of use cases. To this end, we identify essential integration points of the modeled quality properties referring to certain parts of the mashup infrastructure, which are dedicated to perform quality requirements capture and evaluation.

4.1 Quality-Aware Web Mashup Platform

Fig. 3 provides an overview of the quality-aware Web mashup infrastructure, which comprises the following major parts: the device, which allows for user interaction with executed mashup applications, the mashup runtime environment (MRE), the mashup component repository (CoRe) and Web-based resources and services, serving as content providers for mashup components. A mashup application is represented by a composition model specifying layout, screen flow, contained mashup components and their communication. This application model is interpreted by the MRE, which integrates demanded mashup components. According to [18], the described infrastructure is founded on the well-established architecture of CRUISe and EDYRA[2], where each mashup component provides a declarative interface descriptor using the Semantic Mashup Component Description Language (SMCDL). Each mashup component may instantiate the quality property metamodel to reproduce its component-specific quality property values. Thus, quality properties belonging to components may be maintained using the CoRe. Property values are persisted as an extension of SMCDL. Thus, a comprehensive representation of a component including mutable quality properties is achieved. Integration and querying of the application quality properties into the mashup composition model are performed in the same manner.

Mashup components and applications are in the main scope of the presented metamodel. Since user profiles, devices and runtime platforms are also eligible for carrying quality properties, the access methods and modeling conventions hold for them as well. A quality metamodel for mobile device environments is developed in parallel, cf. [19]. It is subject to be integrated in this mashup platform, resulting in extra red bubbles connected to the device and the user in Fig. 3. To add security for model modification, conceptually separated submodels can be merged on demand, depending on the kind of access. If, e. g., the

[2] `http://mmt.inf.tu-dresden.de/cruise`, `http://mmt.inf.tu-dresden.de/edyra`

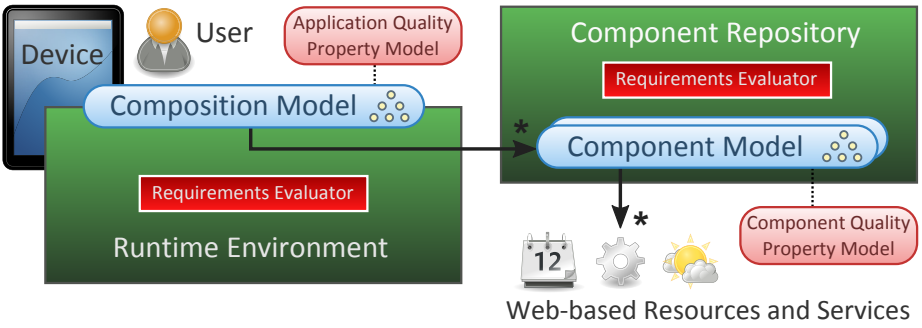

Fig. 3. Quality model integration into the mashup infrastructure

runtime environment or a component browser should display mashup component properties, it can be given access to the whole model. If instead a client is about to modify a quality property, such as adding a rating to the component or setting recently calculated statistical usage data, only the mutable part of the property model is provided through the Web service interface. In this case, the property classification helps to identify runtime-mutable properties.

4.2 Property Capturing and Requirements Evaluation Process

Knowledge of the instantiation location of each metamodel element is not sufficient to run a quality-aware mashup infrastructure. Additionally, we need to distinguish, how and when certain property values are set and updated. There are three kinds of delivering quality property values. First, a component's or an application's author might provide property values, cf. Table 1, which can be edited directly in the corresponding component or composition model or by using a repository frontend. Second, the runtime environment may read context sensors, which are adaptively queried according to the specification in corresponding quality requirements. Context data, such as memory consumption as a measurable quality property, is stored as a value via the repository's Web service and thus persisted into the SMCDL. Third, collectible quality property values such as average user ratings have to be tracked. The repository, CoRe in case of mashup components, is responsible for maintaining this kind of properties.

A major goal of quality property modeling is to evaluate quality requirements, preferably in an automated way. The basic concepts of requirements-driven quality engineering for Web mashups were introduced in [20]. To this end, *requirements evaluators* are designated to perform checks against quality property model instances. In this context, a quality requirement specifies, when or in which temporal intervals it shall be evaluated. Both evaluation at development time, cf. use case ②, and at runtime, cf. use case ③, have to be possible. Thus, requirements evaluators are located at the MRE and at the repository, see Fig. 3, but still the same models can be used. This feature fosters the use of so called end-user development (EUD) tools for user-driven development of applications, which benefit from joint development time and runtime.

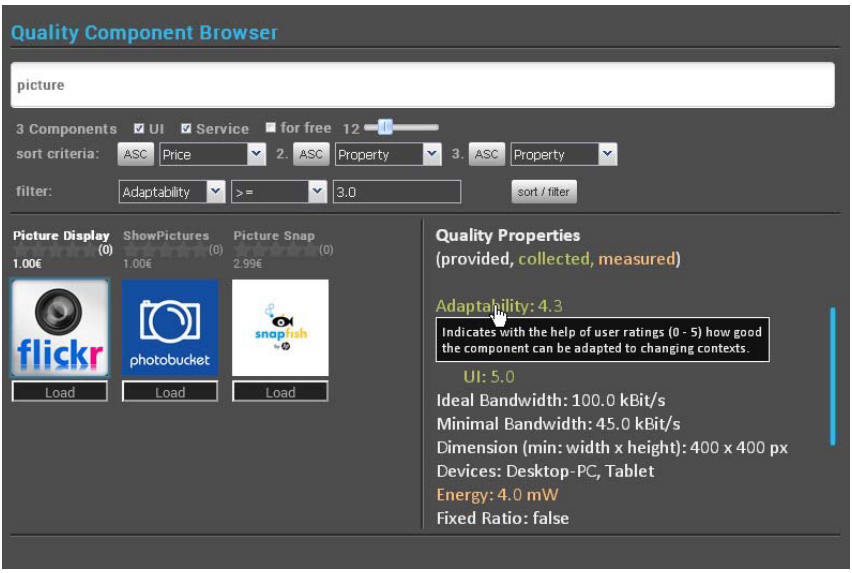

Fig. 4. Browsing mashup components using quality properties

5 Implementation

Beyond the implementation of the quality property metamodel itself, we implemented access and modification interfaces for model instances. Furthermore, a prototypical frontend integration for use case reproduction was built. As indicated in Section 3, the proposed quality property metamodel is implemented using OWL. This brings several advantages like easy modularization or RDF-based querying. In fact, separation of sub-models for components and applications is achieved through model imports. OWL class inheritance is used to distinguish property types. The metamodel implementation integrates well into the existing RDF-based and SMCDL-based mashup modeling infrastructure, including tooling support by means of ontology editors or frameworks. RDF-based quality modeling facilitates SPARQL querying on RDF models, provided by a CoRe Web service. *SPARQL Update* enables model instance modification besides read access. Thus, clients such as a CoRe browser, the MRE or a mashup builder may use this endpoint to manage quality properties. For example, access to the provided properties of a component can be handled with a SPARQL request to the component's OWL representation held in the CoRe, facilitating advanced sorting and filtering by property values. OWL and SMCDL representations are kept synchronized by the repository, making model export continuously available.

For testing quality model instances and the previously outlined property management Web service, we implemented several clients, which are tooling frontends for building Web mashup applications. Beside a CoRe frontend, which is rather an administration tool for browsing and editing quality properties, we integrated a quality component browser into our EUD mashup platform. Fig. 4 shows a

screen shot of this browser, which is also able to process larger composition fragments represented by a composition model. As a client for the CoRe quality property Web service, it provides a convenient tool for use case ①. To this end, an application developer may use quality properties to specify quality requirements in a frontend. Sorting based on multiple criteria, filtering and browsing quality properties of alternative mashup building parts as well as displaying a description of all properties is possible.

6 Conclusion

Although Web mashups started their existence as quick-to-develop situational applications for skilled programmers, their current popularity and advanced platforms demand for regarding a huge spread of quality requirements. Based on a set of use cases, we elucidated the need for mashup-tailored quality properties. Therefore, we proposed a quality property metamodel incorporating well-known Web service NFPs and UI-specific properties. Each property is either provided by author, measured or collectible and has concrete measuring instructions as well as concrete data types, ranges and default values. Specific properties are suitable to be assigned to a special concern such as backend data or functionality.

Introducing a Web mashup infrastructure, we outlined, which architectural components are integration points for quality property model instances. Especially mashup components and application compositions were in the focus of use cases as quality property carrying entities. However, the modular metamodel and its OWL-based implementation allow for easy integration of further sub-models such as platform ontologies or user contexts. Prototypical implementation comprises also a Web service for model instantiation, querying and modification as well as frontends including a quality-aware mashup component browser.

Future work concentrates on integrating platform quality models, which are developed in separate research projects. Moreover, current research includes requirements specification, evaluation and automated invocation of adaptivity actions in different scenarios. To expand the target user group in the field of end-user-driven development of Web mashups, fuzzy specification and configuration of quality requirements is another hot topic on our schedule.

Acknowledgements. The work of Andreas Rümpel is funded by the European Union and by the Free State of Saxony. The work of Vincent Tietz is granted by the European Social Fund (ESF), Free State of Saxony and Saxonia Systems AG (Germany, Dresden) within the project eScience (contract no. 080951807).

References

1. Pietschmann, S., Nestler, T., Daniel, F.: Application Composition at the Presentation Layer: Alternatives and Open Issues. In: Proc. of the 12th Intl. Conf. on Information Integration and Web-based Applications & Services. ACM (2010)
2. Rümpel, A., Radeck, C., Blichmann, G., Lorz, A., Meißner, K.: Towards Do-It-Yourself Development of Composite Web Applications. In: Proc. of the Intl. Conf. on Internet Technologies & Society 2011, pp. 231–235. IADIS Press (2011)

3. Tietz, V., Rümpel, A., Liebing, C., Meißner, K.: Towards Requirements Engineering for Mashups: State of the Art and Research Challenges. In: Proc. of the 7th Intl. Conf. on Internet and Web Applications and Services. XPS (2012)
4. International Organization for Standardization, ISO/IEC IS 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models (2011)
5. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. Concept and Definitions of Software Quality, vol. I (1977)
6. Grady, R.B., Caswell, D.L.: Software Metrics: Establishing a Company-Wide Program. Prentice Hall (1987)
7. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1, 77–106 (2005)
8. O'Sullivan, J., Edmond, D., ter Hofstede, A.H.M.: Formal description of non-functional service properties. Queensland University of Technology (2005), http://www.wsmo.org/papers/OSullivanTR2005.pdf
9. Orehovacki, T.: Proposal for a set of quality attributes relevant for Web 2.0 application success. In: 2010 32nd International Conference on Information Technology Interfaces (ITI), pp. 319–326 (2010)
10. Olsina, L., Mich, L., Sassano, R.: Specifying quality requirements for the Web 2.0 applications. In: ICWE 2008: International Conference on Web Engineering: Workshop Proceedings, pp. 56–62. Slovak University of Technology, Bratislava (2008) ISBN: 9788022728997; Proceedings of: 7th International Workshop on Web-oriented Software Technology, New York, July 14-15 (2008), http://ceur-ws.org/Vol-445
11. Olsina, L., Lew, P., Dieser, A., Rivera, B.: Updating Quality Models for Evaluating New Generation Web Applications. Jnl. of Web Engineering 11(3), 209–246 (2012)
12. Cappiello, C., Daniel, F., Matera, M.: A Quality Model for Mashup Components. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 236–250. Springer, Heidelberg (2009)
13. Cappiello, C., Daniel, F., Koschmider, A., Matera, M., Picozzi, M.: A Quality Model for Mashups. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 137–151. Springer, Heidelberg (2011)
14. Lee, K., Jeon, J., Lee, W., Jeong, S.-H., Park, S.-W.: QoS for Web Services: Requirements and Possible Approaches (2003), http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/
15. International Organization for Standardization, ISO/IEC IS 9126:2001: Software engineering - Product quality (2001)
16. Papaioannou, I., Tsesmetzis, D., Roussaki, I., Anagnostou, M.: A QoS ontology language for Web-services. In: 20th Intl. Conf. on Advanced Information Networking and Applications (AINA 2006), vol. 1, pp. 101–106. IEEE (2006)
17. Ran, S.: A model for web services discovery with QoS. ACM SIGecom Exchanges 4(1), 1–10 (2003)
18. Pietschmann, S., Radeck, C., Meißner, K.: Semantics-Based Discovery, Selection and Mediation for Presentation-Oriented Mashups. In: Proceedings of the 5th International Workshop on Web APIs and Service Mashups, ACM ICPS. ACM (2011)
19. Tietz, V., Mroß, O., Rümpel, A., Radeck, C., Meißner, K.: A Requirements Model for Composite and Distributed Web Mashups. In: Proc. of the 8th Intl. Conf. on Internet and Web Applications and Services (ICIW 2013). XPS (2013)
20. Rümpel, A., Meißner, K.: Requirements-Driven Quality Modeling and Evaluation in Web Mashups. In: Proc. of the 8th Intl. Conf. on the Quality of Information and Communications Technology (QUATIC 2012), pp. 319–322. IEEE (2012)