2012 Eighth International Conference on the Quality of Information and Communications Technology

Requirements-Driven Quality Modeling and Evaluation in Web Mashups

Andreas Rümpel and Klaus Meißner
Faculty of Computer Science
Technische Universität Dresden, 01062 Dresden
Email: {andreas.ruempel,klaus.meissner}@tu-dresden.de

***Abstract*—Mashups are evolving into a leading paradigm for application development using Web-based services. Despite the existence of composition platforms being able to fundamentally match functional requirements, constraint definition in terms of describing quality requirements is very poorly considered so far. Regarding mashup components and applications as well as their runtime platforms and user contexts, existing work is lacking a unified modeling approach for heterogeneous quality requirements and their subjacent properties such as device parameters or user ratings of components. In this paper, we investigate methods of specification and evaluation of quality requirements in Web mashups. Therefore, we propose a concept for modeling mashup quality requirements and properties, considering requirements sources, targets and involved stakeholders. A characteristic set of penalty actions for non-fulfillment of requirements is discussed as a part of the evaluation infrastructure.**

I. Introduction and Motivation

Years ago, mashups of Web-based resources became popular, promising fast application results by bringing together simple building parts. However, the added value was reached by providing aggregated and filtered data [1] or visualization of data. First, only little interaction of the mashed application building parts was possible. Very simple resources types such as feeds or simple REST APIs were used, but later, more complex Web services with extensive interface descriptions (WSDL) served as inputs for Web mashups. Meanwhile, modern composite user interface (UI) mashups are able to produce complex applications by integrating UI building parts in a component-based and service-oriented fashion [2, 3]. While the first simple mashups were intended to be developed very situationally and without explicitly specifying any requirements in advance, the aforementioned modern mashup platforms rely on component and composition models, i. e., functional interface descriptions, which allow at least for basic functional requirements matching. However, they do not offer an infrastructure or process for requirements engineering on custom non-functional properties.

Figure 1 illustrates a very simple Web mashup with two visual mashup components and connected background services. The left component visualizes a Twitter feed and extracts keywords from news update messages periodically. Next, a randomly selected keyword is communicated to the other component, which displays a picture delivered by a keyword-based query result of a bound picture service like Flickr. At this point, imagine a large pool of functionally equivalent components. Comparing component-specific *quality properties*, the application developer could specify *quality requirements* to make the platform select the most suitable one. Since this recommendation or selection scenario is by far not the only use case, an infrastructure for a multitude of different quality requirements is worth striving for. The Web mashup, its building parts and execution context are potential carriers of quality properties and requirements as well. In a multi-device scenario, the mashup application could, e. g., demand special device capabilities to distribute its components according to a specific layout. Promoting conditions like the introduction of workflows and the provision of in-house Web services make Web mashups progressively attractive in business scenarios. To this end, quality properties, such as average service response times or component ratings are of huge commercial importance. Moreover, it is desirable to observe requirements adherence or even force it, e. g., by providing engines and workflows for requirements evaluation and an appropriate set of penalty actions to punish contract violations.

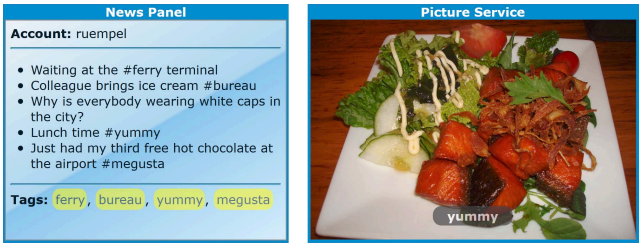

Fig. 1. Simple news visualization Web mashup application

As a main contribution of this paper, we propose a unified semantic modeling approach of properties through over different Web mashup platform parts, such as mashup components, applications, user and developer contexts, runtime environment and devices. Based on those properties, the specification of different kinds of quality requirements is facilitated. Therefore, we discuss requirements targets, intended purposes and stakeholders. Since requirements gain relevance, if they can be automatically evaluated and actions like component exchange and adaptation can be performed, an infrastructure for the evaluation of such quality requirements is introduced.

The remaining paper is structured as follows. Section II outlines existing work in modeling of quality properties and requirements with regard to Web mashups and services. Next,

978-0-7695-4777-0/12 $26.00 © 2012 IEEE
DOI 10.1109/QUATIC.2012.51

a unified quality property landscape is presented in Section III. Based on these properties, Section IV regards sources, targets, formalization and handling of quality requirements for Web mashups. Finally, Section V concludes this paper.

II. Related Work

Developing mashup applications is not a completely new discipline, but has many related ones, e. g., concerning application building in a component-based fashion or Web service composition. In the Web service domain, most approaches for modeling non-functional, i. e., *quality properties*, recruit their property models from the international standard for the evaluation of software quality ISO/IEC 9126 [4] and its successor, the ISO/IEC 25000 series [5]. The former defines functionality, reliability, usability, efficiency, maintainability and portability as relevant characteristics. Beyond their quality property models, Web service quality modeling approaches do not offer user-defined requirements nor automated evaluation nor triggering penalty actions [6]. At least, goal-driven Web service composition is supported by the Web Service Modeling Ontology (WSMO) [7] and OWL-S (service profile) [8]. They define non-functional properties in a quality model, but main implementations are not able do handle those properties. Thus, they only have conceptual character. WSMO defines a model of non-functional properties, which are mainly inspired by the metadata elements of Dublin Core. According to [9], three Web service quality dimensions, those concerning runtime, cost management and security, can be identified. They also reveal, that the "main drawback [...] is the lack of an interoperable language associated to the Web service quality definition." Similar to [10], *penalties* are regarded as a "means for compensating" non-fulfillment of service contracts. Therefore, penalties of real-world scenarios should be considered as suitable actions.

Quality properties for Web services are primarily used for matchmaking and ranking purposes. Beyond this, Web mashups entail a broader range of use cases, because they produce user-suitable applications with more stakeholders in development and execution processes, while composing Web services, e. g., using BPEL, yields new Web services or business processes. Although component-based software systems have commonalities to composite Web mashups, they differ in the granularity of their building parts. Composite Web mashups compose black-box Web resources, while component-based software systems in general allow for deep inspection. Thus, specification and monitoring of non-functional properties can be arbitrarily fine-grained, cf. [11], making their contract languages unsuitable for Web mashups.

Existing *mashup quality models* cover both mashup component properties [12] and properties of the composite application [13]. The latter are calculated by aggregating individual component property values, e. g., by minimum, maximum, average or sum. The regarded quality model decomposes into API quality, data quality and presentation quality. Ratings scale within integer values, e. g., *security operability* evaluates SSL support, the presence of user accounts and API keys,

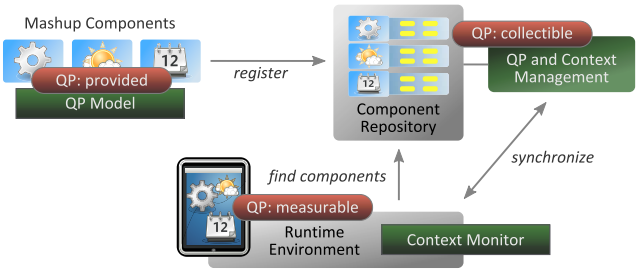

Fig. 2. Sources of Web mashup quality properties

each counting "1 point". However, only property models are presented, assuming a certain optimization direction promoting a quality enhancement. Concurring properties or user-defined requirements on the proposed quality models are not discussed.

According to [14], there is a general lack of requirements modeling in mashup development compared to traditional software engineering. The only workflow for at least functional requirements specification is to match and select mashup components based on their component interface descriptions. Altogether, a *predefined improvement direction* of all properties is assumed, like "a high value *is good for* the quality" of an application or component, cf. [15]. Custom requirements specification is widely neglected. Traditional software constraint languages are only of limited benefit for requirements definition in mashup scenarios, because of pre-authored black-box components, late application changes during runtime and distributed evaluation needs. Furthermore, a *penalty management to encounter non-fulfillment of requirements* is named as a challenge, but concepts are missing.

III. Unified Quality Property Modeling

Since similarities exist especially between Web service interface descriptions (WSDL) and mashup component interface descriptions, cf. SMCDL [16], quality property descriptions will integrate very well at component level. Beside mashup components and the mashups themselves, other carriers of quality properties have to be considered. Quality requirements are intended to be specified as constraints over properties of more than one property carrier. To this end, we propose a unified modeling approach for all of those property carrier types, that our target mashup platform comprises: (1) *mashup component*, (2) composite *mashup application*, (3) *runtime environment* and client *device* parameters and (4) *user and developer contexts* and profiles.

A. Sources of Web Mashup Quality Properties

Based on their sources of specification, we distinguish between three kinds of quality properties in a Web mashup infrastructure like the one in [3]. They are illustrated by the red rounded boxes in Figure 2. First, quality properties can be *provided* by authors of artifacts like mashup components. This applies, e. g., to license information, a statically defined price or a certificate. The definition of this kind of properties is likely to be done within an existing interface description. Since those properties are invariant for a certain temporal scope, their

values can be read out at design time as well as at execution time. Second, properties can be *measured* at runtime. Hence, their values are only accessible while executing a Web mashup and may change dynamically. Examples are current response times of used Web service operations, memory consumption or the user's current location as a context information. The runtime environment is responsible to manage the corresponding sensors. Finally, some quality properties, such as component ratings or average Web service availability values are considered as *collectible*. They have to be stored within a repository or context service. Typically, the base values for collectible properties are measurable inconstant properties.

B. Property Model Distribution and Formalization

An access strategy for model fragments has to be elaborated to facilitate requirements definition over distributed properties. Since mashup platforms are already grounded on a model-based description of components and compositions, we recommend the specification of quality properties together with pre-existing descriptions of the subjected entity. Unfortunately, most mashup platforms do not offer dedicated property descriptions of their runtime environments. User and developer profiles containing, e. g., a history of used and developed applications, favorite mashup components and statistical data, are regarded as *context information*. Typically, user context specifications already make use of semantic technologies for the specification of their context models and instance data. Obviously, communication and addressing properties over models and APIs with different technologies is very heterogeneous. We strive instead for homogeneous modeling and addressing of quality properties. Since context models and some mashup platforms use *semantic modeling techniques* for domain knowledge such as RDF and OWL, we propose these to be used to describe the properties, non-functional and functional, of all carriers. Thus, a quality requirement can also reference functional properties without conceptual overhead.

IV. Quality Requirements and Actions

Assuming a unified property landscape such as outlined above, this section describes, which kinds of quality requirements are common to be used in a mashup environment. We first describe a requirement's structure and possible directions, i. e., sources and targets. Then, we propose possibilities of formalizing such requirements and a set of actions to be invoked as penalties in case of a "contract infringement".

A. Quality Requirements Structure and Relations

As illustrated in Figure 3, quality requirements reference property instances. In fact, the main part of the requirement defines *constraints* over property values. Fortunately, it is possible to model both non-functional and functional properties for all relevant property carriers in the same way using semantic modeling languages, cf. Section III. Properties and instance data form a distributed *mashup property model*, which ca be accessed by requirements evaluation engines. Entities carrying properties are mashup components, developers, users,

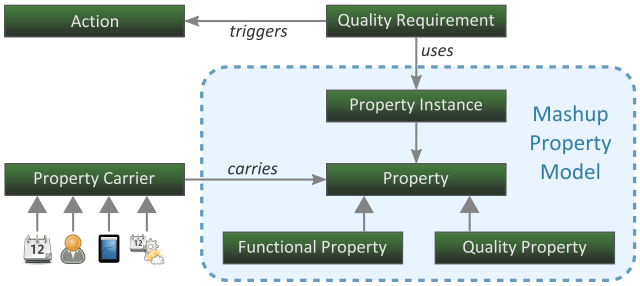

Fig. 3. Mashup quality requirements metamodel

devices, runtime environments and the composite application. In case of a constraint infringement, the requirements author can define, what the *penalty* should look like by assigning adequate actions. Moreover, a requirement contains information about its evaluation time, e. g., immediately at the time of system entry, specification or loading of the application, once after a specified delay or periodically timed.

B. Requirements Sources and Targets

Beside dedicated requirements engineers, different infrastructural components of a mashup platform can themselves be requirements sources or subjects. Figure 4 shows, that some directions of specifying quality requirements are more likely than others. Green arrows indicate reasonable relations highlighted by an example or purpose. Instead, red arrows mark rather uncommon requirements directions. Since mashup components should be exchangeable within mashup applications, the requirements between those two entities are unidirectional, e. g., for selection purposes based on quality properties or runtime exchange. This kind of task can also be expressed by the developer or user at development time. In this case, the requirement would be evaluated by an engine in the component repository, cf. Figure 2. User contexts can also be requirements subjects, e. g., when requesting credentials for a certain background Web service of a mashup component. Regarding the runtime environment and device properties, only incoming edges have common use cases, such as mashup components requiring device-specific adaptivity or applications, which distribute their building parts in a multi-device scenario.

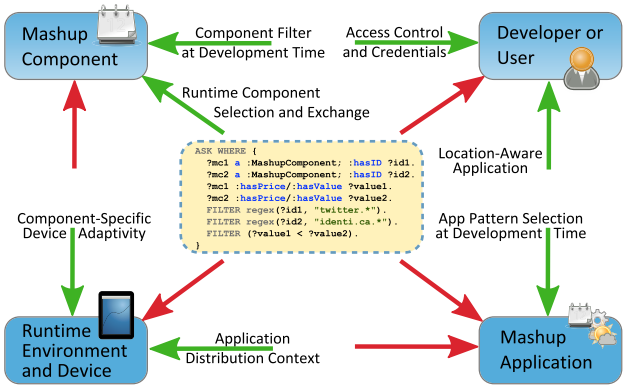

Fig. 4. Directions of quality requirements and SPARQL sample requirement

C. Requirements Formalization and Evaluation

A constraint definition over properties is the core part of our quality requirements. Since we recommend the description of properties using semantic modeling languages, such constraints can use semantic query languages like SPARQL, which is able to operate on RDF-based knowledge representations. The middle of Figure 4 shows a possible representation of a comparative requirement, which queries price values of two mashup components. To this end, it is assumed, that price and id information is defined in a corresponding property model, which is provided by each mashup component, cf. Figure 3. Based on this *ASK* query result, *true* or *false*, the requirements evaluation engine decides, whether to initiate a specific action. If the price of the *twitter* component is evaluated as greater than the price of the *identi.ca* component, the requirements author could have planned to exchange the previously used twitter component by an alternative as an adequate action to punish this constraint violation. This yields the following requirements evaluation workflow. (1) SPARQL constraints are extracted from the requirements descriptions. (2) The evaluation engine, which is located at the runtime environment and the component repository, parses the intended execution times and conditions. (3) The evaluation engine executes a query like the one of Figure 4, which represents a requirements constraint. (4) This constraint may refer to a number of properties, which are influencing the query result. (5) Based on this result, a set of actions is invoked.

D. Actions and Penalties

Among the existing work considering quality requirements, only few papers recognize the necessity for penalties. Inspired from real-world scenarios, we hold that penalty actions are an effective way to enforce requirements compliance. Especially in Web mashup scenarios, options to react to constraint violations are manifold. Beside feedback actions such as user or developer notifications or interaction requests, the composition model could be modified by rewiring, adding or removing mashup components. Also the triggering of component internal adaptation or instance replacement are useful reactions. Moreover, the runtime environment could load an alternative application configuration (template), switch to a different quality profile or let components migrate to other devices or between client and server, when using a distributed mashup platform. Most of the presented actions permit two options of execution: *automated* and *recommended*. While automated action execution does not require user interaction, the final decision is made by the user in case of utilizing the recommendation approach. Of course, actions need parameters, e. g., telling, which mashup component should be replaced and which is the replacing component implementation.

V. Conclusion

Since Web mashups are increasing in their complexity, the need for specifying quality requirements is getting crucial. Especially in enterprise use of mashups, evaluation, monitoring and reacting to non-functional properties and requirements is indispensable. Our proposal is founded on a quality property model powered by semantic modeling languages. To this end, a homogeneous property modeling is achieved for all relevant entities in a Web mashup platform such as components, the application itself, user and developer contexts and device parameters as well as the runtime environment. Based on this property landscape, the definition of quality requirements is facilitated not only for a user, but also for other requirements sources, which we identified. As the main part of a quality requirement, constraints refer to a certain set of properties and decide, whether penalty actions have to be invoked. Such actions range from component-internal adaptation to user ratings of Web mashup building parts. We argue, that using our proposed quality requirements and an appropriate evaluation workflow allows for a more precisely defined quality perception and ensures requirements enforcement through penalty actions in scenarios incorporating modern Web mashups.

References

[1] Yahoo! Inc., *Pipes*, 2012. [Online]. Available: http://pipes.yahoo.com/.

[2] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan, "Hosted Universal Composition: Models, Languages and Infrastructure in mashArt," in *Conceptual Modeling – ER 2009*, ser. Lecture Notes in Computer Science, vol. 5829, Springer, Nov. 2009, pp. 428–443.

[3] S. Pietschmann, M. Voigt, A. Rümpel, and K. Meißner, "CRUISe: Composition of Rich User Interface Services," in *Proceedings of the 9th International Conference on Web Engineering (ICWE 2009)*, Springer, Jun. 2009, pp. 473–476.

[4] International Organization for Standardization, *ISO/IEC IS 9126-1: Software engineering – Product quality – Part 1: Quality model*, 2001.

[5] ——, *ISO/IEC IS 25000: Software engineering – Software product Quality Requirements and Evaluation (SQuaRE)*, Aug. 2005.

[6] M. Thirumaran, P. Dhavachelvan, S. Abarna, and G. Aranganayagi, "Architecture for Evaluating Web Service QoS Parameters using Agents," *International Journal of Computer Applications*, vol. 10, no. 4, pp. 15–21, Nov. 2010.

[7] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, pp. 77–106, Nov. 2005.

[8] W3C, *OWL-S: Semantic Markup for Web Services*, Nov. 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/.

[9] M. Comuzzi and B. Pernici, "A framework for QoS-based Web service contracting," *ACM Trans. on the Web*, vol. 3, no. 3, pp. 1–52, Jun. 2009.

[10] J. O'Sullivan, D. Edmond, and A. H. M. ter Hofstede, "Formal description of non-functional service properties," Queensland University of Technology, Brisbane, Tech. Rep., Feb. 2005. [Online]. Available: http://www.wsmo.org/papers/OSullivanTR2005.pdf.

[11] S. Röttger and S. Zschaler, "Tool Support for Refinement of Non-functional Specifications," *Software and Systems Modelling Journal*, vol. 6, no. 2, pp. 185–204, 2007.

[12] C. Cappiello, F. Daniel, and M. Matera, "A Quality Model for Mashup Components," in *Web Engineering*, ser. Lecture Notes in Computer Science, vol. 5648, Springer, Jun. 2009, pp. 236–250.

[13] M. Picozzi, M. Rodolfi, C. Cappiello, and M. Matera, "Quality-based Recommendations for Mashup Composition," in *Web Engineering*, ser. LNCS, vol. 6385, Springer, Jul. 2010, pp. 360–371.

[14] V. Tietz, A. Rümpel, C. Liebing, and K. Meißner, "Towards Requirements Engineering for Mashups: State of the Art and Research Challenges," in *Proceedings of the 7th International Conference on Internet and Web Applications and Services (ICIW 2012)*, Xpert Publishing Services, May 2012.

[15] P. Lew and L. Olsina, "Instantiating Web Quality Models in a Purposeful Way," in *Web Engineering*, ser. Lecture Notes in Computer Science, vol. 6757, Springer, Jun. 2011, pp. 214–227.

[16] S. Pietschmann, C. Radeck, and K. Meißner, "Semantics-based discovery, selection and mediation for presentation-oriented mashups," in *Proceedings of the 5th International Workshop on Web APIs and Service Mashups*, ser. ACM ICPS, ACM, Sep. 2011.