Towards Requirements Engineering for Mashups: State of the Art and Research Challenges

Vincent Tietz, Andreas Rümpel, Christian Liebing, and Klaus Meißner
Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany
{*vincent.tietz,andreas.ruempel,christian.liebing,klaus.meissner*}@*tu-dresden.de*

***Abstract*—Mashups have become popular in the modern Web providing a lightweight development approach for mainly small and situational applications. Visual composition metaphors and loosely coupled widgets encourage the fast implementation of changing requirements. However, domain experts and mashup composers are poorly supported in expressing and formalizing their needs, leading to time-consuming and error-prone component discovery and composition. Methods and techniques of traditional requirements engineering (RE) are not transferable out of the box due to targeted user groups, isolated development phases, insufficient tool support, and different models. Therefore, we investigate characteristics of software engineering in mashup approaches compared to similar development paradigms revealing and discussing challenges when applying RE to mashups.**

***Keywords*-Web mashups; requirements engineering; software development process; application modeling; UI composition**

I. Introduction

Presentation-oriented *mashups* introduce the user interface (UI) as a new integration layer for component-based applications. They have become a prominent approach for the lightweight integration of distributed and decoupled Web resources. Originally, mashups have been developed by script-based assembly of heterogeneous application programming interfaces (APIs). However, incorporating UI fragments, the heterogeneity of mashup building parts and the composition effort increases. Thus, more powerful mashups are possible to be built at the expense of a simple development approach. Since there is no widely accepted understanding about a general mashup development process, also the manifestation of RE remains uncertain. Beside available work regarding traditional, component-based and Web-based RE, at least the necessity of a structured development process for enterprise mashups is recognized [1].

Mashups promise less expensive and faster application development of long tail applications integrating existing components and services customized by users. Apart from very simple, situational mashups, the explicit specification of functional and non-functional application requirements is essential. Driving factors are the reuse of pre-existing business processes or task models (cf. [2]), the growing application complexity, and business plans of service providers considering mainly non-functional requirements. However, the requirements elicitation and specification in mashups is neglected so far, making the quality-aware discovery and integration of components difficult. Moreover, missing model-based development approaches impede platform independence, adaptation, reusability, and maintainability.

To identify research challenges when applying RE to mashups in Section IV, we initially introduce the principles of RE, its core activities, and the characteristics of the application type *Web mashup* in Section II. Selected approaches are evaluated based on a catalog of criteria defined in Section III. Finally, Section V concludes this paper.

II. Principles

This section gives a brief overview of the principles of RE, the related core activities, and techniques. Further, we introduce Web mashups as our target application type.

A. Requirements Engineering

Software RE is the process of discovering the purpose of a software system by identifying and documenting stakeholders and their needs in order to achieve a satisfying software system for which it was intended [3]. Therefore, RE is multidisciplinary and human-centered, whereas the communication of requirements (implying readability, validity, and comprehensibility) and the management of requirements (implying traceability, searchability, and changeability) are critical success factors. The *development process* is considered as the instance of a process model defining *roles*, *artifacts*, and *activities*. The *technique* defines how to perform an activity, while the *method* combines both activities and techniques.

Traditionally, the first activity in RE is a *feasibility study* often coupled with a *risk analysis*. If the project effort is estimated to be adequate, the *elicitation and analysis* phase follows. There are a plenty of techniques available for requirements elicitation, such as *interviewing*, *brainstorming*, *analyzing existing documentation*, *data mining*, and *prototyping* [3]–[5]. In general, these techniques can be applied to different kinds of development processes. They are rather independent of the target application type applying social, psychological, and analytic strategies.

In contrast, the next activity, *elaboration and specification* of requirements, is intended to reveal the requirements in order to specify functional and non-functional product descriptions for subsequent development activities. The objective of structured requirements elicitation and specification is to achieve completeness and consistency, covering all requirements, and getting non-ambiguous specifications. The specified requirements constitute a contract, which forms the basis for later *validation* and iteratively making compromises within the *negotiation phase*. The *validation phase* is based on previously specified requirements. Depending on the domain and type of a software product, requirements change over time. Typically, *requirements change management* cares about additional or belated requirements or changes in previously taken decisions, also regarding potentially increasing costs or development effort.

The formalization degree of requirements artifacts affects the degree of automation for refinement or validation, e. g., using automated test cases, model checking or monitoring. To increase transferability and reusability, modeling techniques have been established to specify requirements in a formal way. One widely used modeling facility for object-oriented software is the Unified Modeling Language (UML). Formal specifications enable the refinement and transformation of models covering separate concerns of the development process and the final product [6]. The chance for model-driven refinement of computation-independent requirement models to technological independent conceptual models in the design phase is one major incentive for the application of model-driven RE. Ideally, the generation of executable code from these models leads to low-error software products fulfilling the specified requirements.

B. Web Mashups

Mashups evolved from simple data-driven aggregation of feeds to complex applications composing Web- and UI-based building parts. In general, tools like *Yahoo! Pipes*[1], *JackBe Presto*[2], and *IBM Mashup Center*[3] support the visual composition of technology-independent Web services, APIs, and UI components by dragging components on a canvas and wiring output and input channels in order to create new applications. Compared to other software systems, mashups are rather small applications with only few stakeholders, such as component developer, mashup composer, and mashup user. Regarding methods, patterns, and tools, simplicity and reusability are important demands, since mashup development by users with low programming skills, also referred to as *end user development*, is getting popular.

In general, mashup components can be considered as self-contained entities solving user tasks. Figure 1 shows an example Web mashup, which allows planning of a conference participation. In order to receive suggestions for routes of the public transportation services, a participant needs to define start and destination locations as well as corresponding temporal constraints. In addition, the user requires information about available hotels and the weather near the conference location. Therefore, the task *plan conference participation* is decomposed into *get travel info*, *get weather*, and *find hotel*. These tasks are supported by appropriate components (e. g., a map) that need to be selected and composed.

[1]http://pipes.yahoo.com/
[2]http://jackbe.com/products/
[3]http://ibm.com/software/info/mashup-center/

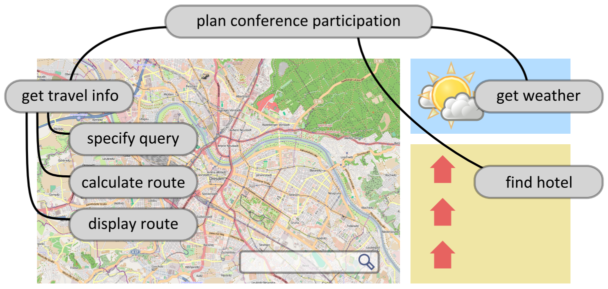

Figure 1. Conference participation mashup with tasks

Despite the simplicity of composition metaphors in current tools, the component discovery remains difficult. Searching is occasionally facilitated by keywords, interface descriptions, and community feedback. However, in the light of growing repositories and ambiguous tags, the identification of proper search criteria becomes an increasing challenge for inexperienced users. Further, mapping of component interfaces, e. g., originating from different providers, that should communicate within the same application, is not trivial. Thus, we argue that this should be addressed by a model-driven and semantics-based development approach.

III. State of the Art

To support RE, a wide range of development processes, methods, and tools have emerged. In the following, we especially evaluate model-driven development approaches for Web applications to review the state of the art and to reveal research challenges when applying RE to mashups.

A. Evaluation Criteria

As a foundation for our evaluation criteria, we adopted the evaluation framework in [7] for a survey of RE in Web- and service-based development approaches, facing mashup-relevant aspects regarding *requirements model*, component-aware *development process*, *model-driven development*, and *adequate tooling support*.

1) Requirements Model: Requirements, which should be provided by a software, are specified in a requirements model. Commonly, they are divided into non-functional, functional, and domain-related requirements [3]. *Functional requirements* define functions of a software system or parts of it. They are intended to accomplish calculations, data manipulation, or other processing consisting of inputs, behavior, and outputs. From the user's point of view, mashups and their constituent components support the user in solving his or her specific tasks. Technical details and system characteristics are usually hidden by black-box components externalizing their functionality by their interfaces only. Therefore, we evaluate how functions (FR1), their sequence (FR2), their input and output (FR3), and the relationships of outputs and inputs (FR4) are specified.

Functional requirements are supported by *non-functional* requirements (NFRs), or *quality requirements*, which impose constraints on the design or implementation [3]. Product quality criteria in common development processes are mainly guided by the international standards for the evaluation of software quality ISO/IEC 9126 and ISO/IEC 25000. Internal and external metrics specify different quality criteria such as availability, efficiency, or security, defined in a quality model. If regarded, they are usually specified in poorly structured documents making it very hard to reuse them or automatically check and monitor their violations. Since NFRs imply a destination or *association point*, cf. [8], we evaluate, whether the method allows the specification of related objects in a suitable granularity. Such associated objects can be functional requirements, system artifacts such as components, resources, or the project context (NFR1). Further, specification possibilities of how a NFR is scaled or measured within the target system is evaluated as *integrability* or *operationalization* (NFR2).

Software is developed and used within an application domain. This includes, for example, the organization's structure, business rules, goals, tasks and responsibilities of its members, and the data that is needed, generated, and manipulated [3]. A comprehensive *domain model* provides an abstract description of the world in which an envisioned system will operate. It provides also knowledge structures and therefore allows reasoning on facts, as well as opportunities for reuse within a domain [3]. Because the domain model is an integral part of any requirements specification, we evaluate how domain models are supported describing data structures (DM1), roles and stakeholders (DM2) and how existing domain models can be reused (DM3).

2) Development Process: The development process needs to be adopted to the target user group in order to provide a lightweight and efficient development approach, wherein the knowledge about available components needs to be considered in the requirements phase. If requirements are identified at an early stage and components are chosen at a later stage, there is a bigger chance that components do not support the required features [9]. The main advantage of applying RE to mashups is that requirements can be matched to predefined components [10] and to support the development process by prototype generation, providing instant feedback. Therefore, we evaluate how the requirements phase is embedded in the whole development process, including the support of round-trip engineering (DP1) to automate model synchronization. Since we consider composite mashups, we evaluate whether the development process is component-based (DP2) and whether component recommendation is supported (DP3). Finally, we evaluate how the decomposition of requirements is supported (DP4). This also applies to NFRs, which are likely to be decomposed into NFRs or FRs.

3) Model-driven Development: We argue that model-driven development (MDD) [6] needs to be applied in order to benefit from traceability, usability, and platform independence in mashups [11]. Therefore, we evaluate existing work considering the provision of precise metamodels (MD1) and mappings and transformations into conceptual models based on the requirements model (MD2). Moreover, this implies some kind of component discovery (MD3) that may be facilitated by the use of semantic knowledge (MD4) [12].

4) Tool Support: Since mashup development is rather user- and prototype-driven, adequate tools are needed to support requirements elicitation and specification. Regarding the tool support we evaluate the provision of visual requirements modeling (TS1) and of managing development artifacts (TS2). We further try to identify the target user group of these tools to decide whether they are applicable to mashup composers (TS3) and how they are guided through the development process. To this end, we specified the following user groups: requirements engineer (RE), modeling expert (ME), software developer (SD), and end user (EU).

Since we regard model-driven development as a key demand and mashups as Web-based applications, we focus on corresponding development methods in our evaluation. Initially, we discuss traditional software engineering, as it provides the most comprehensive and mature approaches for RE and MDD. Finally, we consider current mashup approaches illustrating the lack of RE support.

B. Traditional methods and techniques

The notion *traditional software development* is used here to describe mature and well elaborated methods and techniques usually applied in industrial software projects and object-oriented development processes. Amongst industrial practitioners, the requirements elicitation phase is most often realized with the help of scenarios and use cases followed by focus groups and informal modeling [4]. Since object-oriented analysis is the most popular modeling method, we consider Rational Unified Process (RUP) [13] in conjunction with UML and related development tools, such as IBM Rational Software Architect (RSA)[4], as one representative of traditional software development methods.

1) Requirements Model: Regarding the requirements model, we observe that all functional (FR1–FR4) and non-functional requirements can be expressed with the help of use cases and textual supplements. Role assignments are

[4]http://www-01.ibm.com/software/awdtools/systemarchitect/

supported within the *NFR questionnaire* (NFR1) that can be enriched by textual impact descriptions (NFR2). Domain requirements (DM1) are described with the help of use cases and class diagrams. Stakeholder analysis is also supported by business analysis models (DM2). Principally, reuse of domain models is achieved by importing existing class diagrams. However, domain models tend to be application specific. Established repositories or standardized schemas as well as semantic mappings are usually not applied.

2) Development Process: RUP provides the disciplines business modeling, requirements, analysis and design, implementation, test, and deployment that all pass iteratively the phases inception, elaboration, construction, and transition. Within the business modeling discipline the system context is analyzed to capture structure and dynamics of the organization. Business requirements are captured by use case and analysis models that structure information about the organization and the relations to external stakeholders. In the requirements discipline, functional and non-functional requirements are refined by business use case models to system use cases and class diagrams. As round-trip engineering and the usage of component-based architectures are promoted practices in RUP we consider DP1 and DP2 as supported. However, the focus of round-trip engineering is on UML and Java. Components correspond rather to object-oriented artifacts that tend to be tightly coupled. The recommendation of components (DP3) based on the requirements analysis is not explicitly proposed. The decomposition of requirements (DP4) is partially supported by use cases, documents, and interactive refinement, because use cases are not intended for detailed functional decomposition.

3) Model-Driven Development: RUP provides no specific guidance on how to apply MDD, but offers an appropriate basis for it [14]. Although, the usually applied UML use cases provide a graphical notation, the metamodel is limited (MD1). Many aspects such as linking use cases by pre- and post-condition relations are not supported [15]. This leads to the use of text-based templates that do not provide formal precision, tend to be ambiguous and impede automated model transformations. Therefore, the transformation from computation independent model (CIM) to platform independent model (PIM) is limited (MD3). In general, the automatic discovery of components based on semantic knowledge (MD4) is not supported.

4) Tool Support: RSA is one example for the application of RUP for model-driven and object-oriented development. Since it provides visual modeling for use cases and domain models, we consider TS1 as supported. However, the tool is intended to be used by requirement engineers and modeling experts that should be experienced in using UML (TS3). IBM Rational Requirements Composer[5] is proposed to bring all stakeholders together for eliciting and managing requirements (TS2). Supported techniques comprise documents, storyboards, process diagrams, and use cases. Resulting use cases can be integrated in RSA providing traceability throughout the development process. However, detailed formal representations are rarely used and informal representations such as natural languages are still preferred.

[5]http://www-01.ibm.com/software/awdtools/rrc/

C. Web Engineering Methods

In the past, the Web engineering community proposed several model-driven development methods. Prominent examples are OOHDM [16], OOWS [17], UWE [18], WSDM [19], and WebML [20]. The main purpose is the explicit support of Web-specific concerns such as navigation, presentation and personalization by providing conceptual models, and the generation of code. Although, the focus is mainly on the design phase, it is recognized that requirements analysis and specification need to be considered [5, 7].

1) Requirements Model: Regarding requirements analysis and specification usually existing techniques are adopted. For example, the functional requirements of OOHDM, UWE, and WebML are represented by use cases whereas functions (FR1), their sequence (FR2), input and output (FR3), and their relations (FR4) are mainly defined with the help of text-based templates. WSDM and OOWS propose textual templates and task descriptions such as ConcurTask-Tree (CTT) notation implying a lack of semantic clarity. Data requirements (DM1) are not explicitly supported by OOHDM, UWE, and WSDM [7]. Only OOWS uses information templates and WebML uses a data dictionary and entity relationship models to support data requirements explicitly. With the help of use cases or task models, all methods cover some kind of role specification (DM2). However, the reuse of existing models (DM3) is limited, because the development processes start with use cases leading to new and application-specific domain models.

2) Development Process: Regarding the development process, all methods apply some kind of requirements, design and generation phases, whereas round-trip engineering is not supported (DP1). The design phase is usually divided into conceptual, navigational, and functional modeling. Only OOHDM, its extension OOH4RIA [21], OOWS, and WebML are partly component-based (DP2) by using functionality that is provided by Web services. However, component recommendation is not available in any method (DP3). Web applications are usually generated with the help of templates or manually selected components. Further, the requirements decomposition (DP4) is completely supported in OOWS by using task trees and otherwise supported by use cases or textual descriptions.

3) Model-Driven Development: From the model-driven point of view (MD1–MD2), the specification of application requirements is not considered adequately in order to enable model-based transformations into conceptual models, because traditional Web engineering methods, except OOWS and UWE, do not provide a metamodel for requirements analysis [7]. Therefore, the automatic transformation to subsequent artifacts is not available. Since there is no consistent use of components, their discovery (MD3) using semantic knowledge (MD4) is not supported sufficiently.

4) Tool Support: OOHDM has been supported by HyperDE[6], but the development seems to be discontinued since 2009. In principal, UWE can be used in any tool supporting UML stereotypes, but there is also the specialized tool MagicUWE[7] available. WebML is supported by WebRatio[8] that allows business process modeling, application modeling, generation, and deployment. OOWS provides a CASE tool based on the Eclipse platform [7]. There is no dedicated tool for WSDM available, however CTTE[9] can be applied for task modeling. Overall, only WebRatio and MagicUWE appear to be maintained continuously. Tool support for managing created models or components (TS2) is currently not available. Regarding the target user group (TS3), users need to have knowledge about the associated requirement and modeling techniques. Therefore, model-driven Web engineering tools require similar skills as in traditional methods.

D. Service-Based Engineering

Building applications upon Web services provides simplification of application development by reducing the need for specific code. However, the black-box character of services impedes the prediction of the whole application behavior, which makes quality handling difficult [9] and leads to dependencies on vendors and less flexibilities in requirements. Various approaches for service composition at technical data integration level, e. g., BPEL, have been proposed, whereby the composition is realized using structured programming constructs without considering the interaction with users. To integrate human tasks, BPEL4People has been introduced. However, the problem of service composition at presentation layer is still not supported adequately. The most prominent approaches in this context are ServFace [22], MARIA [23], Achilleos et al. [24], and Tsai et al. [25].

1) Requirements Model: In general, these approaches do not support specific requirements modeling (FR1–NFR2), since their development is achieved by modeling directly on functional interfaces. They are represented by visual service front ends, generated dynamically. Only MARIA supports some kind of RE by task modeling in order to specify functions (FR1), their sequence (FR2), and input and output (FR3) including their relations (FR4). Analogous to Web engineering approaches, service-based engineering does not support entities and relations (DM1) explicitly. In MARIA, roles and stakeholders (DM2) can be associated with different task trees. Finally, the reuse of existing models (DM3) is partly supported in MARIA, while in other approaches the development process starts from the scratch at any time.

2) Development Process: The development process provides mainly design and generation phases. Round-trip engineering (DP1) is not supported, since the approaches are based on only one application model or apply sequential processes, whereby the models involved cannot be synchronized. All approaches are partly component-based due to the use of Web services. Because in general the UIs are generated dynamically, only Tsai et al. support UI services (DP2) and provide recommendation (DP3). Finally, except MARIA that uses task models, requirements decomposition (DP4) is not supported since RE is missing at all.

3) Model-Driven Development: Due to the overall lack of requirements models (MD1), all approaches do not support the transformation of user requirements into conceptual models (MD2). Solely MARIA uses CTTs to specify requirements in a first step, which are transformed into conceptual models. Regarding the component discovery (MD3), all approaches provide Web service repositories to allow a simple search by keywords. Since the approach of Tsai et al. is based on UI services, the corresponding registry supports discovery using semantic knowledge (MD4).

4) Tool Support: Regarding the tool support (TS1) and target user group (TS2), ServFace offers a visual and Web-based authoring tool[10] mainly for end users. MARIAE[11] supports task modeling in the context of MARIA and is mainly intended for requirements engineers. The other two approaches also address application developers with appropriate skills, whereby only the tool of Achilleos et al. provides visual modeling. The management of artifacts (TS2) is limited to Web service components and solely in Tsai et al. UI services may be managed within a repository.

E. Mashup Development Methods

Originally, mashups have been developed by manual, script-based integration of heterogeneous APIs. Addressing non-programmers, mashup tools like *Yahoo! Pipes* and *IBM Mashup Center* have emerged to support the visual composition of technology-independent Web services, APIs, and UI components. Since these tools do not provide model-based composition, we consider model-based integration platforms, providing component and composition models such as mashArt [26] and CRUISe [27].

1) Requirements Model: In general, there is no explicit requirements model in mashups available. Usually, the mashup composer is able to search for components by keywords or other criteria without being supported explicitly in expressing requirements. At the level of implementation, composition models or integration templates can be

[6]http://www.tecweb.inf.puc-rio.br/hyperde/wiki
[7]http://uwe.pst.ifi.lmu.de/toolMagicUWE.html
[8]http://www.webratio.com
[9]http://giove.isti.cnr.it/ctte.html
[10]http://www.servface.org/index.php?view=article&id=117
[11]http://giove.isti.cnr.it/tools/MARIAE/home

considered as requirements for automated matching purposes. Thus, functional specification is only achieved by manually selecting suitable mashup components using their interface descriptions (FR1, FR3). Sequences and workflow-structuring elements can be modeled using application-internal communication paradigms (FR2, FR4). However, this blends into design and implementation activities and does not produce user requirements directly (FR1–FR4). Although first mashup-specific quality modeling approaches exist [28] and prototypically extend mashArt, non-functional *user requirements* specification is not supported (NFR1, NFR2). Data structures (DM1) are predefined and limited to mashup component interface type definitions. Typically, a mashup application is made for one specific consumer role or end user, thus yielding to few variability in roles (DM2). The tripartite mashup role model applies mashup component developer, mashup composer, and end user. In existing mashup approaches, prevalent models, such as business processes, task models, or domain models cannot be reused as input for mashup development (DM3), but would be possible by adding model transformation engines to provide richer input facilities.

2) Development Process: By adding and rewiring new mashup components, a composition can be easily extended or changed, allowing evolutionary development. However, round-trip engineering is not supported due to single application models. The existing approaches are fully component-based (DP2), whereby in CRUISe and mashArt a recommendation mechanism (DP3) supports the integration of suitable mashup components. Using flat hierarchies, requirements can only be matched to component capabilities or the application logic provided by composing them. Thus, requirements decomposition is very limited (DP4).

3) Model-Driven Development: In CRUISe, formalized domain models are used by semantically enriched mashup component descriptions [12] (MD1). However, the model-based requirement specification and derivation of conceptual models is neglected so far. Different model transformations (MD2) are available in CRUISe, but there is no transformation or mapping of user requirements to conceptual models. Similar approaches are not able to cover them as well. Discovery (MD4) is supported via a component repository making use of semantic component descriptions [12]. It is used for interface query for component implementations facilitating integration and exchange (MD3).

4) Tool support: Visual composition tooling is currently not supported in CRUISe, but mashArt provides a user-scoped *mashArt editor* [26] (TS1). Beside the management of mashup components in a repository, included Web-based resources are managed via URI addressing and may be heterogeneously hosted. There is no unified facility of hosting composite applications (TS2). At least people with medium modeling skills and knowledge of component communication paradigms are required throughout all model-based mashup development approaches without appropriate visual tooling support (TS3). To this end, model-based mashup development platforms require advanced modeling skills, while providing fast application results by selecting pre-existing mashup components.

IV. Discussion and Research Challenges

The results of our evaluation are summarized in Table I, whereas it is obvious that all mashup approaches do not consider any kind of RE. In fact, the mashup composer is still constrained to decompose requirements mentally or with the help of other methods. This impedes component recommendation and composition based on user requirements. Apart from that, model-driven Web engineering methods propose several techniques, such as use cases, text templates, and task trees to document requirements. However, over 30 % of practitioners state that they do not model requirements at all [4]. The avoidance of formal representations during the requirements specification phase emphasizes their opinion, that the formalization effort does not pan out. Therefore, we argue that specialized RE for mashups is needed. Based on this, we identified the following main research challenges to apply RE to mashups:

1) How can essential mashup-specific requirements be sufficiently described by formal models? The requirements for mashups differ from the requirements in other development processes, because mashups are built upon components at a characteristic level of granularity and incorporate user interface parameters. In contrast to existing methods and techniques, semantic clarity needs to be achieved in order to enable model-driven and recommendation-based development support. This implies the shift of pragmatic mashup composition to semantic mashup composition and the incorporation of ontologies. While quality requirements are covered in traditional RE methods mainly in textual form, they are hardly observed in mashup approaches. At least, many non-functional requirements with great importance in distributed mashup application scenarios, such as performance constraints or communication security restrictions, can be formalized and measured very well. Therefore, NFRs should be part of the requirements model, specified together with violation consequences, making use of formalized model connections. Also, workflow aspects need to be integrated in such a requirements model to support more complex scenarios in enterprise mashups.

2) How can a requirements model support the composition phase? In fact, mashup development is currently reduced to the design and implementation phase. However, having appropriate requirements models available, the composition can be significantly improved by applying MDD and requirements-based component recommendation. The benefit of MDD for traditional software systems is relatively low, because the refinement steps are still performed manually. In contrast, mashups allow for aligning specified requirements

Table I
REQUIREMENTS ENGINEERING FOR MASHUPS AND RELATED DEVELOPMENT PARADIGMS: EVALUATION RESULTS

		RUP	OOHDM	OOWS	UWE	WSDM	WebML	OOH4RIA	Servface	MARIA	Achilleos	Tsai	CRUISe	mashArt
Requirements Model														
FR1	Functions	Use cases, Scenarios	Use cases, Text templates	Task charact. templates	Functional use cases	Natural language, Task models	Use cases, Text templates	No	No	Task models	No	No	No	No
FR2	Sequence of functions	Scenarios, Activity diagrams	Use cases, Text templates	Activity diagrams	Activity diagrams	Natural language, Task models	Activity diagrams	No	No	Task models	No	No	No	No
FR3	Input and output	Scenarios, Use cases (Pre- and post-conditions)	Use cases (Pre- and post-conditions), Text templates	Task characterization templates	Use cases (Pre- and post-conditions)	Natural language, Task models	Use cases (Pre- and post-conditions)	No	No	Task models	No	No	No	No
FR4	Relation between output and input	Scenarios, Activity diagrams	Use cases (Pre- and post-conditions), Text templates	Task characterization templates	Use cases (Pre- and post-conditions)	Natural language, Task models	Use cases (Pre- and post-conditions)	No	No	Task models	No	No	No	No
NFR1	Assoc. point model.	Questionnaire	No	No	No	No	No	No	No	No	No	No	No	No
NFR2	Operationalization	Partly (Textual impact descr.)	No	No	No	No	No	No	No	No	No	No	No	No
DM1	Entities and relations	UML class diagram, Business object model, Glossary	Information types	Information Templates	Not explicitly (Content elements)	Not explicitly (Textual descriptions)	Data Dictionairy, Text Templates	Domain models	Not explicitly	Partially (No domain modelling)	Not explicitly	Not explicitly	Predefined by component interface	Predefined by component interface
DM2	Roles and stakeholders	Use cases, Scenarios, Documents	Access capabilities	Task Descr., Interaction Points	Not explicitly (Roles in use cases)	Audience Modeling and Classification	User groups	No	No	Collaboration Tree	No	No	No	No
DM3	Reuse of existing models	Yes	Partly (Design Patterns)	No	No	No	Partly (BP models)	Partly (Domain model)	No	Partly (Task model)	Partly (PML model)	No	No	No
Development Process														
DP1	Round-trip engineering	Partly (UML, Java)	No	No	No	No	No	No	One model	No	No	No	One model	One model
DP2	Component-based development	Partly (Object-oriented)	Partly (Object-oriented)	Partly (Service widgets)	No	Partly (Service model)	Partly (By extension)	Yes	Partly (Web services)	Partly (Web services)	Partly (Web services)	Yes	Yes	Yes
DP3	Component recommendation	No	No	No	No	No	No	No	Partly (UI widgets)	No	No	Yes (UI services)	Yes	Yes
DP4	Requirements decomposition	Partly (Use cases)	Partly (Use cases)	Task trees	Partly (Use cases)	Partly (Textual descriptions)	Partly (Use cases)	No	No	Task trees	No	No	No	No
Model-Driven Development														
MD1	Precise req. metamodel	Partly (Use cases)	Partly (Use cases)	Yes	Yes (WebRE)	No	Partly	No	No	No	No	No	No	No
MD2	Mappings and transformations	Partly (Guidelines)	No	Yes	Yes	No	Partly (Guidelines)	Yes	Yes	Yes	Yes	Yes	No	No
MD3	Component discovery	No	No	No	No	No	No	No	Web service repository	Web service repository	No	UI service repository	Component repository	Component repository
MD4	Usage of semantics	No	No	No	No	No	No	No	No	No	No	UI ontology	SMCDL	No
Tool Support														
TS1	Visual modeling	Yes	No	Yes	Yes	Partly	Yes	Yes	Yes	Yes	Yes	No	No	Yes
TS2	Artifact management	Yes	No	No	No	No	No	No	No	No	No	Yes	Components	Components
TS3	Target user group	RE, ME	RE, ME	RE	RE, ME	RE	RE, ME	ME	EU	RE, ME	ME	SD	ME	EU

with those pre-specified in components. However, since mashups are usually fast developed for situational needs, further development steps need to be robust concerning inconsistent and incomplete requirements. Additionally, round-trip engineering is a necessary prerequisite to make model-driven application development practicable that is currently not provided in any Web-based engineering approach.

3) How can the mashup composer be supported in identifying and formalizing his or her needs? In general, mashups are intended to be created by end users or domain experts. Therefore, appropriate tools are needed to hide complexity and to guide mashup composers in expressing their requirements. This implies visual modeling of requirements and compositions as well as fast application generation. In fact, tools should provide concrete requirements activities which blend into the existing design activities, e. g., by drawing required tasks and dynamic refinement. At the same time, semantic modeling needs to be supported easily, incorporating established ontologies. Application generation and fast prototyping needs to be provided to support users in reviewing and adjusting their requirements.

V. CONCLUSION

This paper presents an evaluation framework consisting of several criteria to analyze RE in software development paradigms, facing relevant aspects of building *Web mashups*. To this end, we analyzed prominent approaches in detail that may be used to develop our target application type. By applying our criteria, it turned out that each of the development approaches provides a distinctive development method, while none is supported at all in existing mashup platforms. However, RE is crucial for mashup development as well to achieve an efficient and requirements-aware development. To substantiate this, we identified three main research challenges. We propose the specification of requirements in formal models to increase reusability, the establishment of a requirements model to support the composition phase and finally an adequate visual authoring tool that supports the mashup composer in identifying and formalizing his or her needs. Thus, we are convinced that addressing these challenges will yield to a more efficient, user-friendly, and quality-aware mashup development process.

Acknowledgment

The work of Vincent Tietz is funded by the European Social Fund (ESF), Free State of Saxony and Saxonia Systems AG (Germany, Dresden), filed under ESF-080939514. Christian Liebing is funded by the ESF and Free State of Saxony (Germany), filed under ESF-100088145.

References

[1] W. Ketter, M. Banjanin, R. Guikers, and A. Kayser, "Introducing an agile method for enterprise mash-up component development," in *IEEE Conf. on Commerce and Enterprise Computing*, Jul. 2009, pp. 293–300.

[2] V. Tietz, S. Pietschmann, G. Blichmann, K. Meißner, A. Casall, and B. Grams, "Towards task-based development of enterprise mashups," in *Proc. of the 13th Intl. Conf. on Information Integration and Web-based Applications & Services*, Dec. 2011.

[3] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proc. of the Conf. on The Future of Software Engineering*, ser. ICSE '00, 2000, pp. 35–46.

[4] C. Neill and P. Laplante, "Requirements engineering: the state of the practice," *Software, IEEE*, vol. 20, no. 6, pp. 40–45, Nov.–Dec. 2003.

[5] J. Escalona and N. Koch, "Requirements engineering for web applications – a comparative study," *Journal of Web Engineering*, vol. 2, no. 3, pp. 193–212, 2004.

[6] B. Selic, "The pragmatics of model-driven development," *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, 2003.

[7] P. Valderas and V. Pelechano, "A survey of requirements specification in model-driven development of web applications," *ACM Trans. on the Web*, vol. 5, no. 2, pp. 1–51, May 2011.

[8] M. Kassab, O. Ormandjieva, and M. Daneva, *A Metamodel for Tracing Non-functional Requirements*. IEEE Computer Society, Apr. 2009, vol. 7, pp. 687–694.

[9] G. Kotonya, J. Hutchinson, and B. Bloin, *Service-oriented software system engineering: challenges and practices*. Idea Group, 2005, ch. A Method for Formulating and Architecting Component and Service-oriented Systems, pp. 155–181.

[10] V. Tietz, G. Blichmann, S. Pietschmann, and K. Meißner, "Task-based recommendation of mashup components," in *Proc. of the 3rd International Workshop on Lightweight Integration on the Web*. Springer, Jun. 2011.

[11] S. Pietschmann, "A model-driven development process and runtime platform for adaptive composite web applications," *International Journal On Advances in Internet Technology*, vol. 4, 2010.

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

[13] P. Kruchten, *The rational unified process: An introduction*. Pearson Education Limited, 2004.

[14] A. Brown and J. Conallen, "An introduction to model-driven architecture (Part III): How MDA affects the iterative development process," http://www.ibm.com/developerworks/rational/library/may05/brown/, 2005.

[15] G. Génova, J. Llorens, P. Metz, R. Prieto-Díaz, and H. Astudillo, "Open issues in industrial use case modeling," in *UML Modeling Languages and Applications*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3297, pp. 52–61.

[16] D. Schwabe, G. Rossi, and S. D. J. Barbosa, "Systematic hypermedia application design with OOHDM," in *Proc. of the 7th ACM Conf. on Hypertext*, ser. HYPERTEXT '96. New York, NY, USA: ACM, 1996, pp. 116–128.

[17] J. Fons, V. Pelechano, M. Albert, and Óscar Pastor, "Development of web applications from web enhanced conceptual schemas," in *Conceptual Modeling – ER 2003*, ser. Lecture Notes in Computer Science, vol. 2813, 2003, pp. 232–245.

[18] A. Kraus, A. Knapp, and N. Koch, "Model-driven generation of web applications in UWE," in *Proc. of 3rd Intl. Workshop on Model-Driven Web Engineering*, 2007.

[19] O. Troyer and C. J. Leune, "WSDM: a user centered design method for web sites," in *Proc. of the 7th Intl. WWW Conf.*, 1998, pp. 85–94.

[20] S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (WebML): a modeling language for designing web sites," *Comput. Netw.*, vol. 33, pp. 137–157, June 2000.

[21] S. Melia, J. Gomez, S. Perez, and O. Diaz, "A model-driven development for GWT-based rich internet applications with ooh4ria," in *Intl. Conf. on Web Eng.*, 2008, pp. 13–23.

[22] M. Feldmann, T. Nestler, K. Muthmann, U. Jugel, G. Hübsch, and A. Schill, "Overview of an end-user enabled model-driven development approach for interactive applications based on annotated services," *Proc. of the 4th Workshop on Emerging Web Services Technology*, pp. 19–28, 2009.

[23] F. Paternò, C. Santoro, and L. D. Spano, "Exploiting web service annotations in model-based user interface development," in *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, ser. EICS '10. New York, NY, USA: ACM, 2010, pp. 219–224.

[24] A. Achilleos, G. M. Kapitsaki, and G. A. Papadopoulos, "A Model-Driven Framework for Developing Web Service Oriented Applications," in *ICWE'11*, 2011.

[25] W.-T. Tsai, Q. Huang, J. Elston, and Y. Chen, "Service-Oriented User Interface Modeling and Composition," *IEEE Intl. Conf. on e-Business Engineering*, pp. 21–28, 2008.

[26] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan, "Hosted universal composition: Models, languages and infrastructure in mashart," in *Conceptual Modeling – ER 2009*. Springer, 2009, vol. 5829, pp. 428–443.

[27] S. Pietschmann, V. Tietz, J. Reimann, C. Liebing, M. Pohle, and K. Meißner, "A metamodel for context-aware component-based mashup applications," in *Proc. of the 12th Intl. Conf. on Information Integration and Web-based Applications*, 2010.

[28] M. Picozzi, M. Rodolfi, C. Cappiello, and M. Matera, "Quality-based Recommendations for Mashup Composition," in *Proc. of the ComposableWeb Workshop*, 2010.