A Requirements Model for Composite and Distributed Web Mashups

Vincent Tietz, Oliver Mroß, Andreas Rümpel,
Carsten Radeck and Klaus Meißner
Technische Universität Dresden,
Faculty of Computer Science, Germany
{vincent.tietz,oliver.mross,andreas.ruempel,
carsten.radeck,klaus.meissner}@tu-dresden.de

***Abstract*—Mashups have recently become popular due to visual composition metaphors and loosely coupled widgets that encourage the fast implementation of situational web-based applications. However, the mashup development process is still challenging for end-users, since they are compelled to retrieve and combine adequate components for their requirements manually. Therefore, we propose a novel model-based requirements-driven mashup design and composition method. It benefits from the use of semantics-based domain vocabulary accompanying the whole mashup development process. In this paper, we present an ontology for specifying requirements for web mashups, which is suitable for deployment in multi-user and multi-device scenarios. To this end, we employ a distributed multi-user scenario and outline the proposal's benefits in a model-driven web mashup composition process.**

Keywords*—*Web Engineering, Requirements Modeling, Mashup Engineering, Mashups

I. Introduction

Presentation-oriented *mashups* have evolved from simple data-driven aggregation of feeds to complex applications composing Web- and UI-based building parts. Compared to other software systems, mashups are rather small applications providing rich user interfaces. Regarding mashup development, simplicity and re-usability are important demands. Therefore, technical details are usually hidden by black-box components providing declarative externalization of their functionality. However, in many of current mashup development environments, the composition is mainly driven by manual selection of components and low-level assembly by connecting events and operations. With the increasing number of Web-based services and components, the development of applications with current mashup platforms becomes a cumbersome task, especially for end-users [1].

Moreover, in modern application scenarios, the context of use is not limited to a single platform or user [2]. People collaborate via their personal devices in a mobile and ad-hoc manner. To achieve abstraction from platforms and service implementations, there is a need for model-based integration concepts, such as provided by CRUISe [3]. These *composite mashups* provide the basis for realizing platform-independent mashup composition as well as distributed and collaborative scenarios that need to be considered in new development methods. Therefore, we argue that requirements-oriented development is needed to cope with such complex scenarios and to abstract from such composition details [4]. Since, a formal requirements specification for mashups is still not available [1], we propose a task-based requirements model as the foundation for stuctured, semi-automatic mashup development and human-centered design of mashup requirements.

The remaining paper is structured as follows. In Section II, we introduce a collaborative travel planning scenario to motivate the need for the representation of distribution and collaboration requirements. Since we propose an extended task metamodel, related work in the field of task-based modeling is discussed in Section III. In Section IV, we present our requirements model for mashups, and its semantic representation. In Section V, we outline the benefits for the mashup development process. Finally, we discuss our results and provide an overview about the implementation of the requirements model as well as transformation process in Section VI and conclude the paper in Section VII.

II. Collaborative Travel Planning Scenario

To illustrate the modeling approach, we introduce a social travel planning scenario, wherein several participants can synchronously vote for a list of travel offers in a distributed co-located multi-device environment. The result is a shared list of rated offers, which is calculated based on the vote from each planning participant. As illustrated in Figure 1, the abstract task tree consists of two phases: the personal offer research (left) and the collaborative rating phase (right). In the first phase, each planning participant creates a personal list of travel offers. This implies the selection of destination locations, e. g., using a geographical map UI component, and one or more offers provided in this region, e. g., using a list component. For example, a participant could choose a bicycle tour in Ireland or an adventure trip to New Zealand.

In the second phase, each participant shares the personal list of offers with all relevant personal devices (smartphone, tablet PC) and shared devices (Smart TV) in a collaborative setting. Next, each device merges the lists in order to visualize the entries and to enable each participant to rate and rank all offer candidates in a synchronous and collaborative manner. To create a common idea of the ranking result, another subtask of the main ranking task – the shared ranking visualization – is executed in the context of a public Smart TV (shared device in the meeting room). After a participant has ranked an entry of the shared list via his personal device, the ranking is synchronized with the shared ranking visualization using the Smart TV.

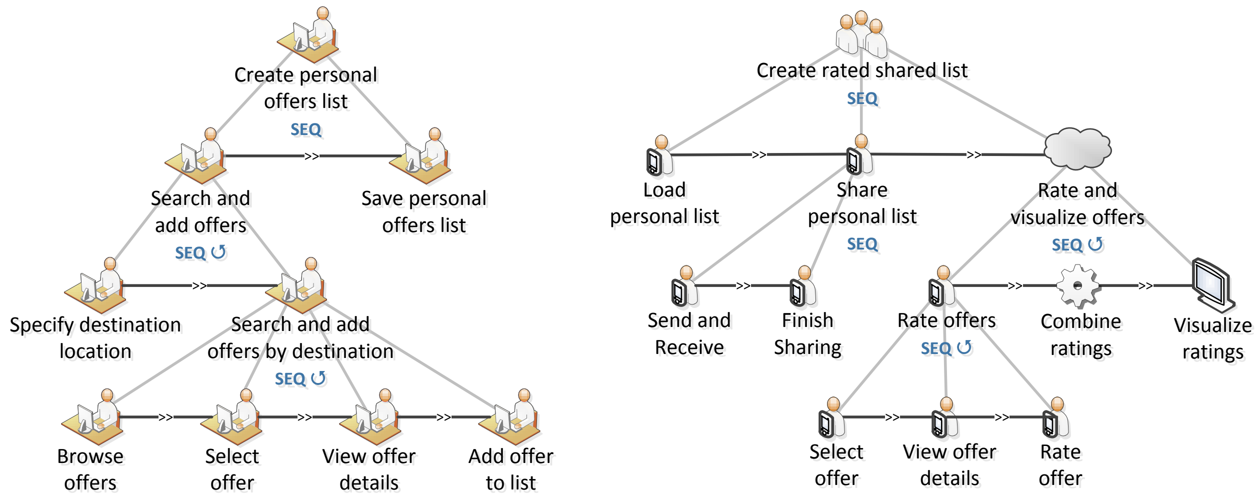

Figure 1. Collaborative multi-device rating scenario

The scenario demonstrates that collaborative tasks are executed in close relation to each other, but in different execution contexts. To create a collaborative application from the task model, additional information and conditions, such as the number of role instances, i. e., collaborators, of the required execution context, is needed. Hence, in the remainder of this paper we present methods to formalize these requirements that we need to determine the binding between a task and its execution context.

III. Related Work

Like in traditional software development, requirements models in the field of Web engineering are, among others, mainly represented by use cases, business process models and task models [5], [6]. Use cases define interaction scenarios between a role and a system in order to achieve a goal, but collaboration and participation among actors cannot be distinguished in UML use case diagrams. Pre- and post-conditions, for example, also need to be added in a textual manner [7]. However, natural language is not suitable to provide a computable requirements model. Further, flow-oriented business process models such as BPMN are well suited for representing workflow requirements. However, both focus on data and system integration and are not suitable to specify individual user requirements [8]. Instead, task models such as ConcurTaskTrees (CTT) [9] are designed for the human-centered specification of user requirements. Because mashup components can be considered as self-contained entities solving user tasks, we argue that task modeling provides both a basis for user-centered requirements modeling and the seamless integration into a model-driven mashup development process.

In the past, several task meta models and notations have emerged with different purposes and degree of formalization and expressiveness. For example, the early Hierarchical Task Analysis (HTA) [10] focuses the hierarchical decomposition of human activities and can be considered as the basis for any subsequent task model. However, tasks are structured by informal plans, goals and actions can be defined on any level of decomposition. GroupWare Task Analysis (GTA) [11] focuses on the collaborative aspect with roles and agents with clear distinction between tasks and actions adopting the activity theory [12]. However, explicit concepts for modeling of distribution are missing. K-MAD [13] is similar to GTA but extends object modeling by supporting abstract and concrete objects. However, their semantics can not be defined. The most prominent task modeling approach is CTT [9] that is used as a starting point in many model-based user interface development (MBUID) approaches. CTT allows the definition of cooperation and communication tasks, which are modeled in collaboration task trees. However, as in all mentioned approaches, distributed and simultaneously performed tasks cannot be expressed explicitly. Finally, there is a lack of semantics and continuous application development, since presentation objects, which are in our case mashup components, need to be selected and integrated manually.

The provision of ontology-based modeling is mainly considered in the field of semantic web services. For example, OWL-S [14] provides an ontology-based specification of web services and templates to initiate automatic search and composition. Similar to that, OWL-T [15] is supposed to support task-based description and discovery. However, the focus is on technical web services with search profiles, grounding facilities, process definitions and result modeling. We adopt some principles such as ontology modeling and semantic matching, but in contrast, we strive for lightweight and user-centered requirements specification focusing on presentation-oriented mashup components.

To generate an application from collaborative task models, several sub models are needed to describe the execution context of the corresponding task. Pribeanu et al. [16] present a hybrid approach to describe context-sensitive and context-insensitive tasks in a single task model. Thereby, CTT is combined with additional model elements such as decision graph nodes and arcs that describe the context of use of each subtask. However, this approach focuses only on how context information can be integrated into the task model, but it does not clarify the context information that is relevant to

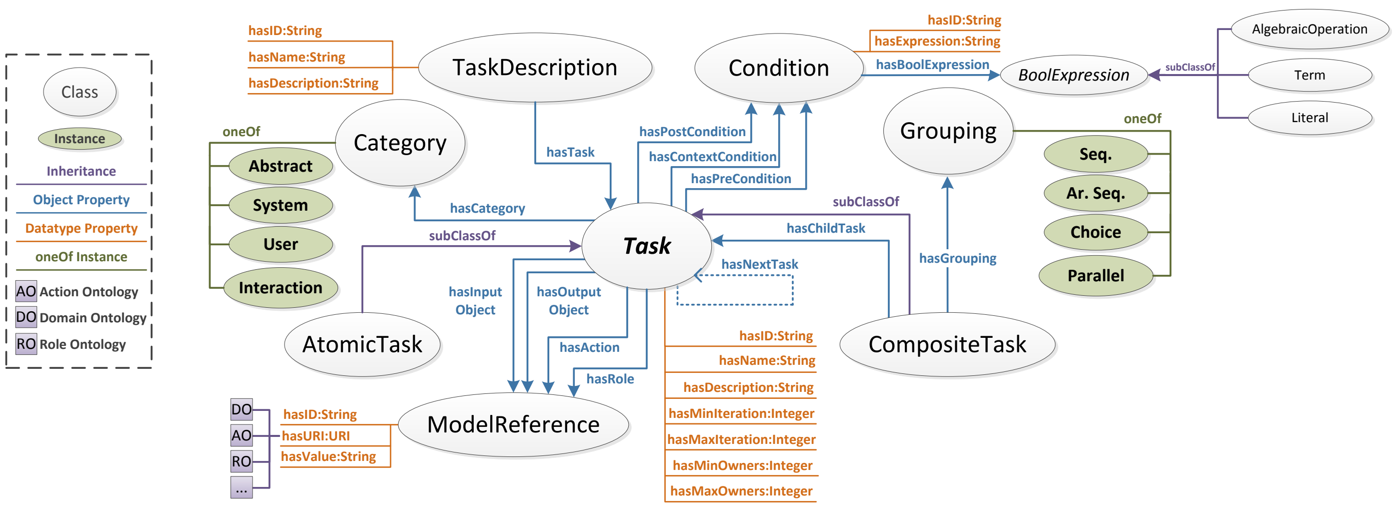

Figure 2. Task ontology

perform context-sensitive tasks in a collaborative fashion. The MBUID approach CODAMOS [17] extends DynaMo-AID by an environment model that classifies each available computing device as an interaction resource with input and output channels. The corresponding environment ontology contains concepts that are referenced by the task model. Finally, so-called modality interaction constraints are used to map tasks to a set of potential interaction resources. However, as in all MBUID approaches relying on the CAMELEON reference framework [18], additional abstract user interface (UI) descriptions are required to transform the task model into a dialog and presentation model. We argue that the mashup paradigm allows us to create web applications with sophisticated UI in a more flexible way. In contrast, we focus on the dynamic mapping of task model entities to black box mashup UI components to reduce the development complexity.

Overall, the lack of degree of formalization regarding data and activity modeling as well as the missing expressiveness regarding the execution context impede the use of traditional requirements modeling approaches in distributed mashups. Hence, we introduce an ontology-based task model in the next section to provide a basis for requirements specification and model-based mappings and transformations.

IV. Requirements Model for Web Mashups

We consider task models as a basis for the specification of mashups because of the intuitive representation of composite user goals and tasks [9], the correlation of tasks and mashup components as tools for solving specific user tasks [19] and the correlation between task models and business processes to represent work structure [20]. Further, we follow [2] by extending task models with context modeling facilities to support collaborative and distributed task design. To show how the mapping from task models to mashup applications works, we refer to Section V. In this section, we describe the main concepts of our task model, which is illustrated in Figure 2 and incorporates such context requirements.

A. *Modeling Activities*

The structuring of human activities is one major goal of task modeling. In general, this is realized by the decomposition of high level tasks to more fine-grained sub tasks. Although the decomposition is provided by all task models, the understanding about the main concepts such as tasks, actions and operations seems to be rather different [12]. Therefore, we propose to apply the activity theory [21] to structure terms and relations in the task modeling domain. Therein, a human activity hierarchy is proposed, consisting of activity, action and operation. An activity is directed to a motive and is performed within a context of environment, other activities or humans. This corresponds to our *Task* concept.

A task, respectively an activity, consists of actions, describing what needs to be performed in order to reach the goal of the activity. In terms of mashups, actions reflect the required functionality of a component to fulfill a specific task. In our task model, this is represented by *Action* that is associated to each *Task*. In contrast to other task models we provide a classification of actions [22] based on literature from the field human-interaction and visualization such as [23], [24] and [25]. According to the systems theory we classify each action as a subclass of *input*, *transform* and *output* actions. A visual representation of a part of this classification is shown in Figure 3. It is notable, that none of the actions is bound to a concrete interaction type or application. Thus, this classification is intended to be an upper ontology for rather domain specific actions. However, all actions can be potentially performed by humans, by components or by both.

Since mashups support interactive and non-interactive tasks, we use the concept *Category* to denote the interaction type, such as in [9]. *System* tasks are exclusively performed by components. For example, "Combine ratings" is modelled as a system task in our previous scenario in Section II. Whereas, *interaction* indicates that an interaction of humans and a UI is required such as "Specify destination location". User tasks require no interaction with the system, e. g., fetching a sheet of paper. An abstract task groups heterogeneous subtasks, e. g., "Rate and visualize offers". Task decomposition is realized by the concept *Composite Task*, which consists of at least two subtasks *(hasChildTask)*. *Atomic Tasks* represent the leaves of the task tree. To define the workflow, respectively the temporal relations between child tasks, we use the concept *Grouping* consisting of *sequence*, *arbitrary sequence*, *choice* and *parallel*. To define the order of tasks in sequences, the following task is defined by *hasNextTask*.

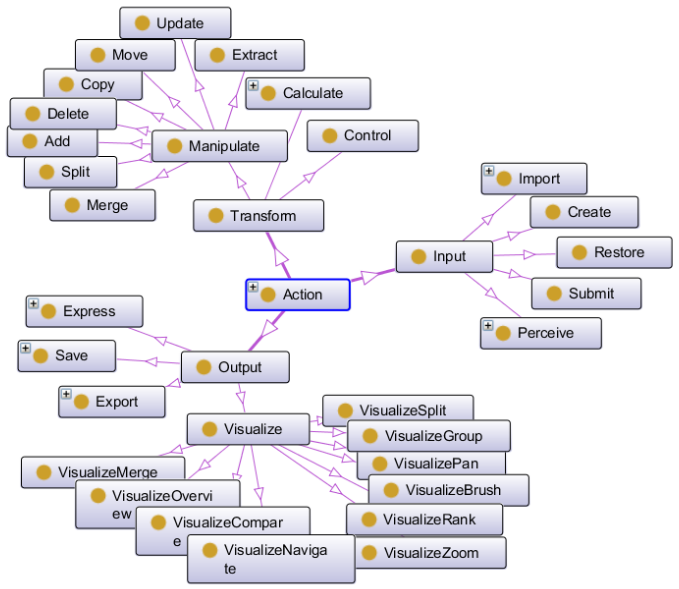

Figure 3. Action ontology

Actions can be very general and usually represent a transformation of a physical or non-physical entity [11]. This is represented by inputs *(hasInputObject)* and outputs *(hasOutputObject)*. The *ModelReference* allows the definition of a data type with the help of a *hasURI*, of a value *(hasValue)* and the instance identification by *hasID*. This also enables the modeling of data flow, wherein output objects of one task can be defined as input objects of another task. In our case, actions, data objects and roles refer to this kind of semantic resources within an action and domain ontology.

Finally, according to the activity theory, operations describe how actions are realized. These reflect the very low-level and modality-dependent interactions of a user such as pressing a key as well as operations of components that need to be performed to provide a needed functionality. However, because we strive for task-based mashup composition and abstraction of implementation details, we do not consider this level of detail in our task model.

B. Modeling Context

To describe the task execution context, we utilize a platform classification that can be used to define context conditions in the task model. It allows the formalization of requirements to perform a task in a specific execution context. We define a platform as a composite of device-, software- and common aspects-specific concepts. To express platform conditions on different abstraction layers we use a vocabulary based on existing classifications such as the *W3C Delivery Context Ontology (DCO)* [26]. Further, we extend the existing modality concept with an additional medium and mode class that are specified as part of the *W3C Extensible MultiModal Annotation Markup Language (EMMA)* [27]. This allows us to characterize the nature of a modality in a detailed fashion, e. g., the description of a *TactileInputModality* with mode *Touch* expresses the necessity of a touch sensitive input device like a multi-touch display or touch-pad.

To allow the description of software-specific requirements in relation to the execution context of a task, the platform model contains corresponding concept elements. The following example will illustrate this. As part of the previous scenario in Section 1, the domain expert defines the atomic task “View offer details” for presenting detail information of a selected travel destination. The UI component should include a video element to visualize the beauty and the richness of the landscape and travel opportunities. To influence the component selection and its distribution (video playing capable device) the domain expert requires a vocabulary to describe the need of the video player capability.

Listing 1 shows an example of a conditional platform statement in SPARQL format [28] to present the expressiveness of our task model. The condition defines constraints for the execution device of a task such as it is capable to play a video. As a consequence, we can use reasoning techniques to improve the quality of the result set during the task-component matching procedure.

```
<owl:NamedIndividual rdf:about="
      sample:MobileDeviceCondition">
    <rdf:type rdf:resource="task:Condition"/>
    <task:hasExpression rdf:datatype="xs:string">
        PREFIX p: <http://example.org/platform#>
        SELECT ?platform
        WHERE {
            ?platform p:executedOn ?device.
            ?platform p:enablesAspect
                 p:BidirectionalCommunication.
            ?device p:embeds ?display; p:isMobile true.
            ?display p:supportsModality ?modality.
            ?modality a p:VisualOutputModality.
            ?modality p:hasMode p:Image; p:hasMode
                 p:Video.
            ?display p:hasResolution p:HVGA.
        }
    </task:hasExpression>
</owl:NamedIndividual>
```

Listing 1. RDF/XML representation of a platform task condition

Besides device- and software-specific concepts, we added a common *aspect* class, e. g., providing a communication classification. In our scenario, the sharing subtask implies a mashup component that should present all traveling planner’s ratings in a visual manner (e. g., a list of shared ratings per each travel opportunity). To execute the shared rating list component, the component’s platform should provide the capability to receive each favorites list and synchronize with updates from other traveling planner’s devices. The collaboration aspect is represented by the role assignment in each task *(hasRole)* and specification how many owners the task might have *(hasMinOnwers and maxOwners)*.

V. Leveraging the Requirements Model

Due to the degree of formalization, the proposed task model is an appropriate basis for further mappings and transformations within a model-driven mashup development process as already outlined in [4]. For this, we rely on a semantic component model as well as a distributed mashup runtime environment. The main concepts for this purpose stem from the model-based mashup composition approach provided by CRUISe [29], [30]. Therein, mashup components and their interfaces are described by the *Semantic Mashup Description Language (SMCDL)* that enables the semantic annotation of component's meta-data and their interfaces. Figure 4 shows the parts of the SMCDL as well as their relations to functional and data semantics that are defined in domain models. This is the foundation for the universal and semantic description of any web resource such as UI widgets or RESTful web services. Therefore, the SMCDL is the basis for the derivation of an executable mashup application from the requirements defined in our task model.

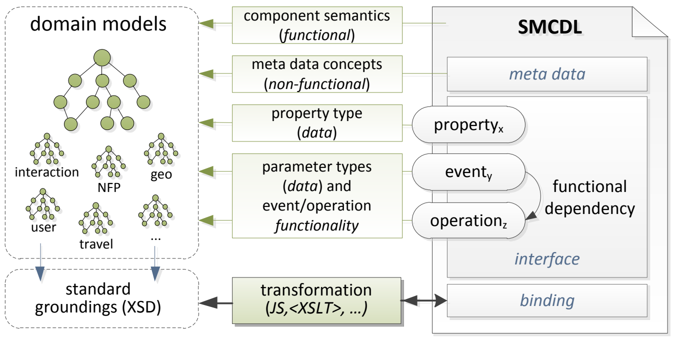
domain models
interaction
NFP
geo
user
travel
...
standard groundings (XSD)
component semantics (functional)
meta data concepts (non-functional)
property type (data)
parameter types (data) and event/operation functionality
transformation (JS,<XSLT>, ...)
SMCDL
meta data
property_x
event_y
operation_z
functional dependency
interface
binding

Figure 4. Semantic annotations used in SMCDL [30]

The transition consists of two parts: (1) the discovery of adequate components for each task in the task tree, and (2) the generation of a composition model that can be executed by the CRUISe runtime. The matching algorithm is already described in detail in [19], and is therefore not in the scope of this paper. However, possible mappings for a part of our travel planning scenario are shown in Figure 5. The matching algorithm returns a set of rated mashup components for each leaf of the task tree. In our case, the task "Specify destination location" is mapped to a map component (a), whereas "Browse offers" (b) and "Select offer" (c) to one list component, since it is possible that one component supports more than one task. Further, the task "View offer details" could require additional components to work such as an image database to show further pictures (d). Finally, it is possible that no component can be found for the tasks in the task tree. Unfortunately, this implies either the re-design of the task model or the implementation of new components.

To build a mashup from the component proposals, we adhere to the platform-independent composition model [29]. The composition model consists of five main parts: (1) the conceptual model that contains mashup components to be integrated, (2) the communication model that interconnects components with the help of communication channels, (3) the layout model that describe the arrangement of components on the screen, (4) the state model (previously called screenflow model) that describes the relation of components and screens, and finally (5) the adaptivity model that describes adaptation aspects. The adaptation model is omitted so far, because adaptation aspects affect only composition aspects such as component's reconfiguration, component mediation and layout

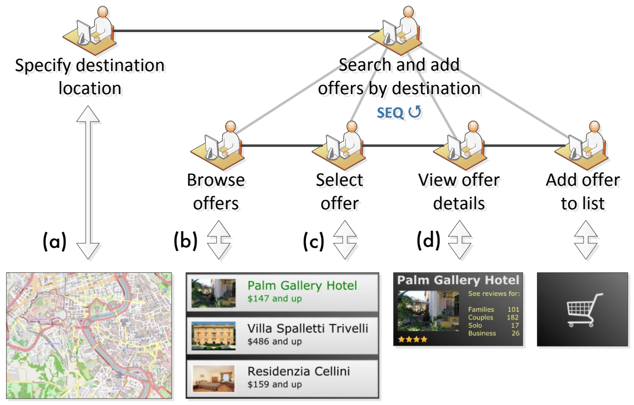
Specify destination location
Search and add offers by destination
SEQ
Browse offers
Select offer
View offer details
Add offer to list
(a)
(b)
(c)
(d)
Palm Gallery Hotel $147 and up
Villa Spalletti Trivelli $486 and up
Residenzia Cellini $159 and up
Palm Gallery Hotel

Figure 5. Mapping of tasks and components

adaptation according to the width and height of the screen. However, the task model is of higher abstraction and is not intended to consider such runtime-specific aspects.

Since, the transformation process cannot be explained in detail here, we just provide a brief overview that is illustrated in Figure 6. Therein, a task model is represented by the tasks $T1$ to $T5$ that are connected by model references, such as *Hotels* as output of $T3$ and input of $T4$, and their grouping, e. g., parallel or sequential. Further, conditions could be defined in the task model. The creation of the composition model is then performed as follows.

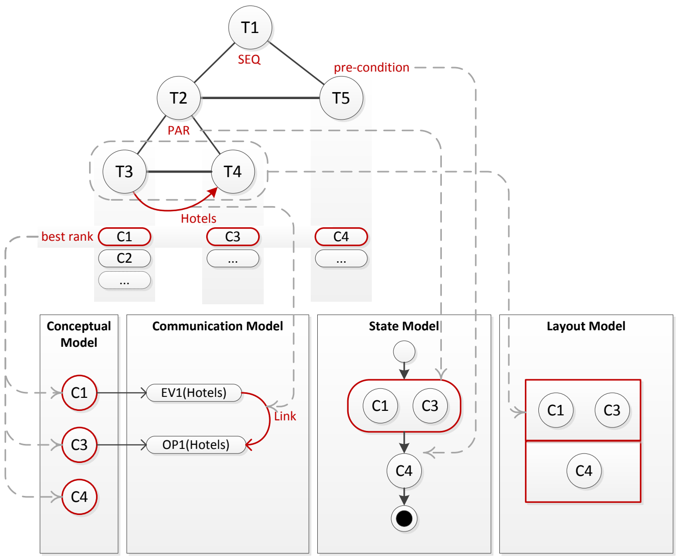

Figure 6. Overview of the transformation process

The conceptual model is created with the help of each component description that is returned by the matching algorithm. Further, the components are reviewed, if they can be connected with each other, i. e., that their semantic descriptions are compatible. This is determined by a matching degree of the parameters of the events and operations. If no matching can be found, it is required to look for further components to make the composition work. For example, additional service components could be needed to fetch some information from external databases. To fill this gap, the semantics-based discovery of CRUISe and mediation techniques [30] are utilized. When the components are identified, they are collected in the conceptual model to build any of the subsequent model parts. In Figure 6 the selected components are $C1$, $C3$ and $C4$.

Based on the previous results of matching of events and operation, the communication model is derived by both the task model and the component descriptions. First, from the correlation of components to tasks and the definitions of input and output references in the task model, we derive which components need to communicate with each other. Second, the annotation of events and operations allows us the mapping to the interface signatures. For example, in Figure 6, the model reference *Hotel* will result in a channel between component $C1$ and $C3$ because $C1$ is related to the task $T3$ which has a model reference to task $T4$. Finally, the component $C3$ is related to $T4$. Therefore, both components should have a common communication channel.

The state model is derived by the grouping attribute of parent tasks and available conditions. For example, the sequence attribute denotes that subsequent components can only be activated if the current component provides the data according to the definition task so that the task can be considered as finished. However, parallel tasks denote that the components are visible and active all the time. For example, $C1$ and $C3$ work in parallel, but since $T5$ is a subsequent task, the related component $C4$ is only activated if $C1$ and $C3$ finished their computation. In fact this means, that related events provide the required data during the mashup runtime.

Finally, the layout model is derived by the allocation of tasks in task tree and the relation of components regarding their tasks. For example, components that correlate with task neighbors are placed nearby on the screen. However, the algorithm can only provide proposals for the layout and the user is afterwards able to configure this as he or she likes. Further, we introduce a card and a tab layout to hide or disable components according to the grouping definition (e. g., sequence, choice and parallel) and the provided conditions in the task tree.

To consider the platform requirements, we extended the composition model by query-based declarative entities (component distribution description). Each define a mapping between the associated component instance and it's platform condition as shown in Listing 1. The quantity of task owners as well as the role definition *(hasRole)* is also considered. However, if necessary, the definition of capabilities per role needs to be done manually after the generation process, since these information is usually not available in the role ontology. The concrete number of distributed component instances is derived from the quantity of task owners that perform a task simultaneously. Details about their communication behavior are derived from the communication model, but can be configured by the composer too, such as the synchronization interval threshold or their communication direction. It depends on the application scenario and can not be discussed in more detail at this point.

When the composition model has been finished by the composer and validated, the application is finally deployed. The model is directly interpreted by a distributed runtime environment that integrates all referenced components from the repository within the execution context of different client-side runtime environments. Further, it handles and coordinates the component's life cycle, communication and their distributed execution. A part of the resulting application (executed in the "personal offer research" according to our scenario in Section II) is shown in Figure 7.

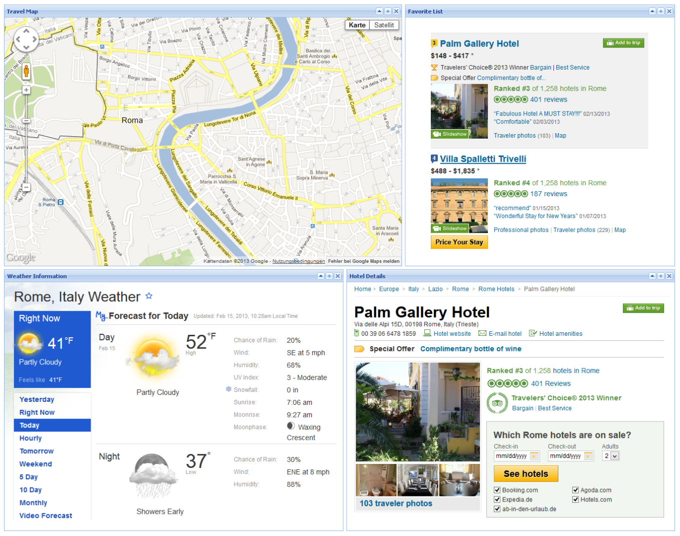

Figure 7. A partial mashup from the scenario

VI. Implementation and Discussion

In this section, we sketch some implementation details of the requirements model and the transformation process, and finally, we discuss our findings.

A. Implementation

1) The Requirements Model: We implemented the requirements ontology in OWL and published an initial version of the proposed meta model on the Web [31]. Listing 2 shows a simple instance of the "Share personal list" task (Figure 1) as an extract in *Terse RDF Triple Language (Turtle)* syntax. The full example is also available in the Web [32]. The first task is an OWL individual of *CompositeTask*, which encompasses two atomic child tasks "Send and recieve" and "Finish sharing" (lines 8-9). The task has an input object that references the semantic concept *favorites list*. Further, the *hasInputObject* relation indicates the count of inbound data channels. Besides the input object, the resulting data of the task is represented by the output object. A context condition is defined in line 10. It refers to the condition that is defined in Listing 1. The collaborative aspect is modeled by the use of the *hasMinOwners* and *hasMaxOwners* properties. The task and its subtasks should be performed in parallel by maximal four and minimal two task owners (travel planners) in the context of their personal mobile devices (lines 16-17).

```
1  :SharePersonalList
2   a activity-tasks:CompositeTask, owl:NamedIndividual ;
3   activity-tasks:hasID "SharePersonalList"^^xsd:string ;
4   activity-tasks:hasName "Share Personal List"^^xsd:string
       ;
5
6   activity-tasks:hasChildTask
```

```
  :FinishSharing ,
  :SendAndReceive ;

activity-tasks:hasContextCondition
  :sample:MobileDeviceCondition ;

activity-tasks:hasGrouping
  activity-tasks:Sequence ;

activity-tasks:hasMaxOwners 4 ;
activity-tasks:hasMinOwners 2 ;

activity-tasks:hasInputObject :Favorites ;
activity-tasks:hasOutputObject :Favorites.
```

Listing 2. Turtle representation of the modeled sharing task

The OWL model is mainly for storing and reasoning purposes. It is obvious that end-users will not create such models without appropriate tool support within the mashup environment. For now, we also created an EMF Ecore [33] representation that allows us to specify a domain specific language (DSL) with the help of EMFText [34]. To provide an example, the same task in Listing 2 is also shown in Listing 3, but using task modeling DSL. The DSL is a more convenient way, and allows the faster creation of text-based task models for testing purposes. Since, the expressiveness required by our task model can be provided by OWL as well as Ecore, the DSL and the OWL representation can be easily transformed into each other.

```
TaskModel TM "Travel planning scenario"

Tasks
    Interaction Task SharePersonalList "Share Personal
         List" {
        action Share
        input Favorites
        output Favorites
        minOwners 2
        maxOwners 4
        Sequence
            Interaction Task FinishSharing "Finish
                 Sharing" {...}
            System Task SendAndReceive "Send and Receive"
                  {...}
    }

References
    Reference Share URI "actions.owl#Share"
    Reference Favorites URI "travel.owl#FavoritesList"
```

Listing 3. The sharing task as DSL

2) The mashup composition process: The instances of the task model are managed by a Java-based *Task Repository (TaRe)* that provides a service interface to store and load task models. It imports RDF/XML as well as DSL-based task models. Further, the TaRe interacts with the CRUISe component repository to find and rate mashup components. The TaRe is also responsible for the generation of mashup composition proposals as described before. Task models are represented internally by an ontology model, facilitating the semantic web framework Jena [35] which allows us to apply SPARQL queries and reasoning techniques.

B. Discussion

As depicted with the previous examples, our task model is applicable to describe simple, more complex and even collaborative scenarios. Besides the travel planning scenario, we also modelled other applications, such as a stock exchange mashup and business trip requests. Further, we evaluated the expressiveness of the task model with the help of the basic workflow patterns [36] and conclude that most of them are supported to model relevant aspects of work. Since we derived the task model from traditional task modeling approaches, we consider that our task model has at least the same expressiveness. Moreover, since the clear semantics provided by the task and action ontology, we are confident of the opportunities for model-driven mashup development. Finally, we showed that a transformation process from a task model requirements specification is feasible.

However, determining the optimal components for single tasks and finding the optimal glue code with respect to interface compatibility and to the task model is of high algorithmic complexity. Although the provided transformation process can produce a draft of the composition model, there are still options and alternatives which might be preferred differently by the other users. Therefore, the user also needs to be able to explore these alternatives and to control the transformation process. Further, composition details such as layout and styles are not covered by the task model explicitly. This is not very surprising as the task model is intended to be an requirements model that abstracts from composition details. Therefore, the user should be able to refine the resulting model manually. To enhance the generation process, we plan to store compositions resulting from a task model and to let users rate them. Based on this, we will be able to generate more detailed composition proposals. Finally, our experience is that the component matching and the composition process is time consuming, depending on the amount of tasks and components. Therefore, we propose to save matching results for later reuse to save calculation time.

VII. Conclusion and Future Work

In this paper, we proposed an ontology-based task metamodel for composite mashups with special aptitude to requirements specification for distributed and collaborative scenarios. To this end, a mashup composer can structure the intended behavior of the application by decomposing corresponding tasks, while simultaneously defining conditions on the distribution, platform or devices. In contrast to other existing approaches, our task models are able to provide clear semantics regarding the actions to be performed within tasks, data objects and conditions. We illustrated the applicability of the presented metamodel by instantiating it for a social decision support scenario.

Future work includes the evaluation of the transition process from a task model to the mashup composition model with the help of a user study. In order to do this, we are currently working on a Web-based task model editor to provide an abstract view on ontology-based task modeling, making it feasible for developers without RDF/XML knowledge or advanced programming skills. Since task modeling activities depend on the application domain, constraints regarding the execution of mashups and their components such as time-out conditions or other quality requirements may be also specified along the task definitions. Conclusively, we claim that task-based mashup development will bring the mashup application building paradigm to a broader audience. Task-based mashup development implies new opportunities for the usage of task knowledge during the composition and runtime, e. g., in order to optimize tasks and dynamic component integration.

Acknowledgement

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. ESF-080951807. The work of Oliver Mroß is granted by the ESF, Free State of Saxony, contract no. ESF-080951831. Same applies for Carsten Radeck who is funded under the contract no. ESF-080951805.

References

[1] 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 (ICIW2012)*, Stuttgart, Germany, 2012.

[2] J. Vanderdonckt, "Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments," *Proc. of XIth Congreso Internacional de Interacción Persona-Ordenador Interacción*, pp. 3–14, 2010.

[3] 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 & Service (iiWAS'10)*, 2010, pp. 413–420.

[4] 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*, 2011.

[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] 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.

[7] 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.

[8] K. Sousa, "Model-driven approach for user interface: business alignment," in *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. New York, NY, USA: ACM, 2009, pp. 325–328.

[9] F. Paternò, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A diagrammatic notation for specifying task models." Chapman & Hall, 1997, pp. 362–369.

[10] J. Annett and K. Duncan, "Task analysis and training design," Hull Univ. (England). Dept. of Psychology., 1967.

[11] M. van Welie, G. C. van der Veer, and A. Eliëns, "An ontology for task world models," in *5th Int. Worksh. on Design, Specification, and Verification of Interactive Systems (DSV-IS)*, 1998.

[12] Q. Limbourg and J. Vanderdonckt, "Comparing task models for user interface design," in *The handbook of task analysis for human-computer interaction*. Lawrence Erlbaum Associates, 2003, pp. 135–154.

[13] S. Caffiau, D. L. Scapin, P. Girard, M. Baron, and F. Jambon, "Increasing the expressive power of task analysis: Systematic comparison and empirical assessment of tool-supported task models," *Interacting with Computers*, vol. 22, no. 6, pp. 569–593, 2010.

[14] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. N. M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. (2004, November) Owl-s: Semantic markup for web services. W3C. [Accessed: 2013-04-16]. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[15] V. X. Tran and H. Tsuji, "Owl-t: An ontology-based task template language for modeling business processes," in *Software Engineering Research, Management Applications, 2007. SERA 2007. 5th ACIS International Conference on*, 2007, pp. 101 –108.

[16] C. Pribeanu, Q. Limbourg, and J. Vanderdonckt, "Task modelling for context-sensitive user interfaces." in *DSV-IS*, ser. Lecture Notes in Computer Science, C. Johnson, Ed., vol. 2220. Springer, 2001, pp. 49–68. [Online]. Available: http://dblp.uni-trier.de/db/conf/dsvis/dsvis2001.html#PribeanuLV01

[17] T. Clerckx, C. Vandervelpen, and K. Coninx, "Task-based design and runtime support for multimodal user interface distribution," in *Engineering Interactive Systems*, ser. LNCS. Springer Berlin / Heidelberg, 2008, vol. 4940, pp. 89–105.

[18] L. Balme, R. Demeure, N. Barralon, J. Coutaz, G. Calvary, and U. J. Fourier, "CAMELEON-RT: A software architecture reference model for distributed, migratable, and plastic user interfaces," in *EUSAI*. Springer-Verlag, 2004, pp. 291–302.

[19] 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.

[20] H. Trætteberg, "Modeling work: Workflow and task modeling," in *CADUI*, 1999.

[21] V. Kaptelinin, *Context and Consciousness: Activity Theory and Human-Computer Interaction*. The MIT Press, 1996, ch. Activity Theory: Implications for Human-Computer Interaction, pp. 103–116.

[22] V. Tietz. (2011, June) Actions Ontology. Faculty of Computer Science, Technische Universität Dresden. [Accessed: 2013-04-16]. [Online]. Available: http://mmt.inf.tu-dresden.de/models/1.11/actions.owl

[23] R. Springmeyer, M. Blattner, and N. Max, "A characterization of the scientific data analysis process," in *Proceedings of the 3rd conference on Visualization'92*. IEEE Computer Society Press, 1992, pp. 235–242.

[24] D. Gotz and M. X. Zhou, "Characterizing users' visual analytic activity for insight provenance," *Information Visualization*, vol. 8, pp. 42–55, 2009.

[25] G. Mori, F. Paternò, and C. Santoro, "CTTE: Support for developing and analyzing task models for interactive system design," *IEEE Trans. Software Eng.*, vol. 28, no. 8, 2002.

[26] R. Lewis and J. M. C. Fonseca, "Delivery context ontology," World Wide Web Consortium, Working Draft WD-dcontology-20080415, April 2008.

[27] M. Johnston, P. Baggia, D. C. Burnett, J. Carter, D. A. Dahl, G. McCobb, and D. Raggett, "EMMA: Extensible MultiModal Annotation Markup Language," W3C Recommendation, 2009. [Online]. Available: http://www.w3.org/TR/emma/

[28] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for rdf," *W3C Recommendation*, vol. 4, pp. 1–106, 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[29] S. Pietschmann, "A model-driven development process and runtime platform for adaptive composite web applications," *Intl. Journal On Advances in Internet Technology (IntTech)*, vol. 4, no. 1, pp. 277–288, 2010.

[30] 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 (Mashups 2011)*. ACM, 2011.

[31] V. Tietz and O. Mroß. (2012, June) Requirements Ontology. Faculty of Computer Science, Technische Universität Dresden. [Accessed: 2013-04-16]. [Online]. Available: http://mmt.inf.tu-dresden.de/models/1.11/requirements-model.owl

[32] O. Mroß. (2012, June) Sample Travel Planning Scenario Ontology. Faculty of Computer Science, Technische Universität Dresden. [Accessed: 2013-04-16]. [Online]. Available: http://mmt.inf.tu-dresden.de/models/1.11/distribution-scenario.owl

[33] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed., E. Gamma, L. Nackman, and J. Wiegand, Eds. Addison-Wesley Professional, 2009.

[34] EMFText: Concrete Syntax Mapper. DevBoost. [Accessed: 2013-04-16]. [Online]. Available: http://www.emftext.org/index.php/EMFText

[35] Apache Jena. The Apache Software Foundation. [Accessed: 2013-04-16]. [Online]. Available: http://jena.apache.org/

[36] N. Russell, A. H. M. T. Hofstede, and N. Mulyar, "Workflow controlflow patterns: A revised view," BPMcenter.org, Tech. Rep., 2006, bPM Center Report BPM-06-22.