The MoSaiC Model and Architecture for Service-Oriented Enterprise Document Mashups

Nelly Schuster[1], Christian Zirpins[1], Mathis Schwuchow[1], Steve Battle[2], Stefan Tai[1]
[1]FZI Forschungszentrum Informatik, Karlsruhe, Germany, [2]Hewlett Packard Laboratories, Bristol, UK
{nschust,zirpins,schwucho,tai}@fzi.de, steve.battle@hp.com

ABSTRACT

Enterprise documents uniquely facilitate organizational collaboration by representing business processes, rules and data in a visual format that can be communicated between collaborators. While IT-supported document collaboration is well established for structured recurring business processes, creative processes that emerge and evolve instantaneously lack an appropriate document collaboration model. Related documents comprise diverse interrelated parts that evolve through collaborative activities of varying participants in an ad hoc manner. As to this, we introduce a novel approach to represent these documents as mashups of services. Document mashups offer an interactive, intuitive and dynamic way to indicate the structure and behavior of document fragments that are provided by human collaborators or IT systems as services. In this paper we present a conceptual model of document service mashups as well as a document service infrastructure and collaboration platform architecture. Our model considers both the structural/layout composition of a document as well as its active behavior to support collaborative relationships. We represent the structural dependencies of document fragments and the collaborative flows of their providers as application-level rules that react to document events. Using our collaboration platform, collaborators declare document layout and interaction rules that are enforced as RESTful service interactions by the underlying document service infrastructure.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Collaborative Computing, Web-based Interaction; H.4.1 [**Information Systems Applications**]: Office Automation - *Groupware, Word processing*

General Terms

Management, Design, Human Factors.

Keywords

Document-Driven Situational Collaboration, Document Service Mashups, Document Service Infrastructure, Document Collaboration Platform.

1. INTRODUCTION: REALIZING OPEN DOCUMENT-CENTRIC COLLABORATION PROCESSES

Enterprise documents provide a means of communicating information in a purpose-optimized (structured, annotated, graphically appealing, legally binding) form of representation in order to share and clarify individual points-of-view with targeted recipients. In the enterprise, documents are often used as an organizational instrument. Sophisticated enterprise documents encapsulate business rules and processes and provide the means for their regulation and enforcement in a highly efficient manner e.g. utilizing customer relationship, and workflow management systems. From a practical perspective, enterprise documents are closely related to organizational collaboration. As organizations are increasingly required to be responsive to planned changes and unplanned incidents, to innovate in the face of these situations and in collaborations with clients and partners, they need to look beyond optimization of recurring business processes and focus on *open collaboration processes*. These processes are situational, weakly structured and highly interactive human-centered collaborations within or across organizational boundaries [13]. Enterprise documents involved in such collaborations are exposed to *ad hoc* changes as new ideas or data from different sources – humans but also systems – come in. Examples can be found for instance in IT service management or software engineering where collaborators have to react on unexpected incidents like bugs or are collaboratively developing solutions which are captured in interrelated documents. Another example for document collaboration is the collaborative composition of a research publication [13]. During the course of the team collaboration, documents evolve to stable results.

Various technologies and tools support collaborative creation and evolution of documents (e.g., CSCW technologies or wikis) or structured workflows or processes (e.g., BPM). However, how to support ad hoc team collaboration within an overall collaborative document evolution process is an open research question. As to this, we introduce a unique approach to integrate service-oriented architecture and enterprise document paradigms.

Our vision of open document-centric collaboration is to put a *service-oriented enterprise document* in the center of an open collaboration process. Representing enterprise documents as compositions of specific document services allows leveraging service-oriented computing technologies [3]. On the one hand, these technologies facilitate the composition of flexible information and workflow systems, integrating lightweight processes, such as collaborative documentation processes. On the other hand, using the emerging mashup paradigm enables the creation of an end-user-driven development environment for service compositions [14]. Participants use the document structure to regulate their activities (who does what and when) in an ad hoc, situational and joint fashion. An electronic document infrastructure needs to enforce these regulations in a non-invasive way by mapping them to an interaction flow between the participants and actively controlling this flow during the collaboration process. Simultaneously, document content represents the state of the collaboration in flow as it integrates the diverse individual outputs of the participating collaborators. During the course of the collaboration process, document content transforms from a record of collaborative conversation towards a publication of stable results.

In order to realize this vision, traditional enterprise document computing has to be extended along two orthogonal dimensions: Firstly, conventional electronic document structure needs to gain flexibility in order for structural definitions to support individuality, partial definition and ad hoc changes of each document instance driven by collaborating participants. We meet this requirement by representing document structure as rule-based mashups of autonomous document fragments that are intended to be specified and maintained by the human collaborators. Secondly, conventional electronic document content needs to gain dynamicity in order for document fragments to reflect and synchronize the ongoing evolution of multiple collaborative activities (e.g. writing, proof-reading and approval of a specific textual building block) that create and transform contents in the course of a collaboration process. We meet this requirement by associating document fragments with a series of interrelated service instances provided by participating collaborators.

In this paper, we introduce a document component model that maps document fragments on software services. The structuring and coordination of services as parts of a growing document is described in a composition model. Based on the document component and composition models, we have developed the architecture of a lightweight document service bus for event-based interaction between document mashups and services. The document service bus provides the infrastructure for a collaboration environment, which allows for end-user-driven ad hoc collaboration in the course of authoring document mashups. In the following, we describe the conceptual and technical foundations of our approach. Section 2 presents our conceptual document component and composition model. Section 3 shows the architectural design of the document service infrastructure and collaboration environment. Section 4 shows an example scenario and use case illustrating our approach. We discuss related work in Section 5. Section 6 concludes with a summary and an outlook on future work.

2. A CONCEPTUAL MODEL FOR DOCUMENT SERVICES MASHUPS

A document service component model and composition model build the conceptual foundation of our approach for service-oriented document-centric collaboration. The component model describes the nature of a document service. Document services provide content creation (e.g., writing text, drawing diagrams, accessing databases) and transformation functions (e.g., proof-reading, translating, layouting), which can be composed into enterprise documents. The structuring and coordination of services as parts of a growing document is described in the composition model that builds on events and rules: each service exposes events indicating state transitions of its underlying resources which are consumed and reacted upon by mashups. Various rules can be defined for such events; e.g., a rule may specify that when a text-based content service emits an update event, a proof-read service is called by the mashup or that at a certain deadline all services that have not yet delivered results are called again. The mashup author is free to specify any rules that fit the collaboration at hand. To increase usability, it is possible to pre-define rule abstractions that represent application-level controls like reminders.

2.1 Document Services

Document services represent activities of collaborators, e.g. creating a contribution, translating text, or proof-reading. The result of an activity is communicated as a *document fragment* which can be included in a document. Services are offered by various sources like systems living in the Inter/intranet as well as human beings.

We have identified different types of such document services resulting in an extensible service type taxonomy. There are two main types, *content source service* and *content transformation service*. On request, a content source service provides a document fragment, i.e., text or other media in any common format. Examples for content source services are human-written paragraphs, table of contents, and pictures provided by services on the Web. Content transformation services primarily transform or work with existing contents, e.g. document fragments or whole documents. We differentiate between layout, publication and functionality services. Layout services transform documents into a format like XHTML or PDF. They require a composed document as input. Publication services send documents to external sources, e.g., post them to a blog, store them on a server or email them to a recipient. Functionality services provide any kind of value-added function, for instance proof-read or language translation, which can be applied to the document or parts of it. As the taxonomy is extensible, new types can be specified or existing ones can be refined.

An activity is time consuming and subject to a common lifecycle: starting with a request, it progresses over various steps until finishing at some point. The evolution of the result is reflected by a Web accessible document fragment. A uniform Web service interface allows to access the resource, i.e., create, read, update or delete it. When a create service request is directed to a service provider, the service provider creates a new service instance in "draft" state that has a unique endpoint. In this state, the document fragment is subject to revisions. As soon as the provider of a service instance considers his collaborative activity as finished, he marks the corresponding document fragment as "final". After an "approval" it will not change anymore.

The evolution of an activity is communicated to some coordinative peers. These peers can trigger lifecycle stages of a document service by means of service requests, for instance requests for creation or update of a document fragment. Service requests carry meta information as well as an input document fragment, if required by the service. The meta information contain the addressed service provider and the service identifier or service type he should deliver. Furthermore, meta information contains the service requester identifier, a timestamp and, in case the request contains a fragment, the format of the fragment and processing information. The document fragment in the request must have a predefined format the requested service can understand. The processing information in the fragment data specifies which part of the content should be used or replaced by the service. Document services communicate their lifecycle changes, for instance revisions in draft state, by means of *document service events*. These events contain a document fragment representing the result of a collaborative activity at that point in time. Additional information specify the event source and date, access restrictions, service type, event type – i.e. updated, deleted, read, created – and auxiliary information, e.g. on how to interpret the fragment. A service provider may offer various types of services. Also, there might be multiple coordinative peers for a single service. For each coordinative peer communicating with a service, a distinct service instance is created which evolves in its own lifecycle.

2.2 Document Service Mashups

A *document service mashup* represents and realizes one coordinative peer for a collaboration process. It coordinates an open set of collaborators providing activities and incorporates their results as parts of a document. A document mashup consists of some *document structure* specifying the layout of document elements, *document elements* as data containers holding activity results, *document services* associated with elements, *collaborators* providing document services, and some *collaboration logic* regulating the flow of collaborative activities represented by document services.

The left box in Figure 1 shows the document composition model, which describes the structural composition of document services to mashups. Each mashup contains exactly one root *document element* that can be *atomic* or *complex*. The nesting of atomic and complex document elements leads to a definition of document layout, i.e. the structural composition. Atomic document elements do not contain other document elements. Complex document elements do so and allow representing a nested document structure.

Document elements hold document fragments that represent the current state of a collaborative activity represented by a document service. This data can be accessed throughout the mashup to express logical relations between different parts.

Each document service in a mashup is associated with a *collaboration role* that stands for a collaborator. The collaboration roles of a mashup are bound to an organizational *participant* or system that provides the associated document service types. This allows for instance to specify that two collaborative activities in a mashup should be conducted by the same participant.

Each atomic document element is associated with at least one content source service. It has a type which equals the type of its content source service. In addition, document elements might reference an unlimited number of content transformation services, e.g., for proof-reading or spell checking. A content transformation service applies to all document elements contained in the document element it is associated with.

The coordination of different document services is specified by collaboration logic. An event paradigm is used to represent relevant situations. Mashups emit events whenever a document element is added, moved or removed. This allows observing structural changes of the document. State changes of document services can be observed by the mashup through events that originate from document services as soon as they are created, updated, or deleted. For instance, changes to a document fragment in a service that are made during the authoring process are propagated as update events. Furthermore, environmental changes can be observed. E.g., platform events are emitted whenever a new document service, mashup, or provider is available or a role is bound to a participant. Changes in time are propagated through timer events. All these events are observed by the mashup and provide the basis to control the associated collaborative flow. Events might also be consumed by document services if they are important for collaborative activities. E.g. the update of one document fragment might trigger the update of another. More generally, events emitted during mashup runtime cause reactions of services and mashups. These reactions are defined in event-condition-action (ECA) rules, which are part of the mashup (see right box in Figure 1). Events are input to these rules; based on the events conditions can be checked and appropriate actions performed, either by services, mashups or by the infrastructure. The *execution* of a mashup thus consists of monitoring events and conditions on these events as well as conducting the activities.

An example rule is "if introduction and summary are delivered, do a proof-read". This is a typical interaction rule that involves different collaboration partners, i.e. the authors of introduction and summary and the proof-reader who is notified as soon as introduction and summary emit an update event and are in state "final". Another rule would be "insert a table of content into the mashup as soon as at least two chapters are provided". This rule has a

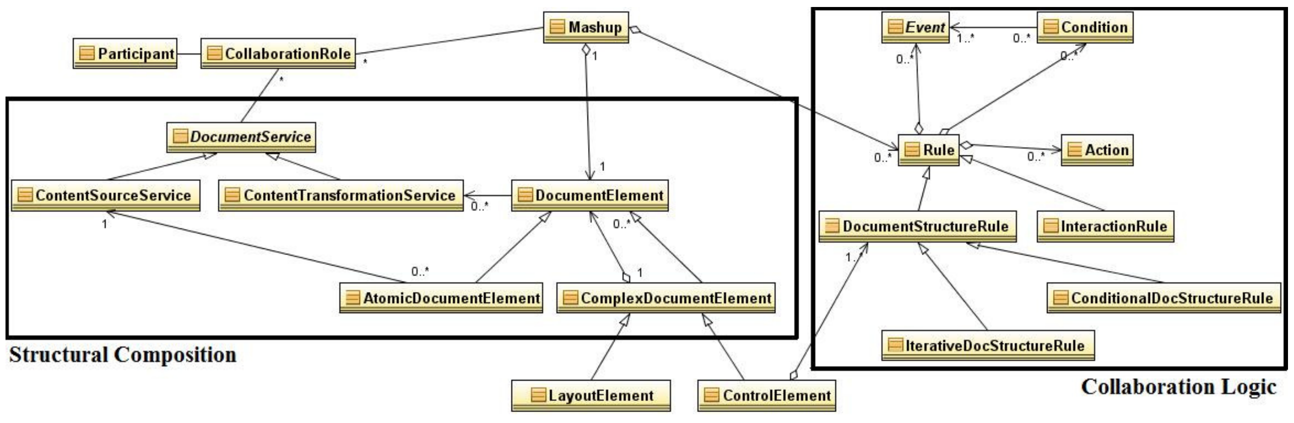

Figure 1. Document Composition Model

more structural character. Generally, we distinguish between *interaction rules*, which involve the collaboration of participants and request the change of a document service state, and *document structure rules*, which do changes to the structure, i.e. the document elements and layout of the mashup.

The document composition model includes a set of fundamental rules that are used implicitly to define default mashup behavior. For instance, whenever a document element binds a new document service, the service receives an update request.

Furthermore, there exist rules, which directly apply to a certain document element, for instance "insert a table of content into the mashup as soon as at least two chapters are provided" refers to a table of content document element. Thus, this *conditional rule* is directly associated with this element in the mashup. An *iterative rule* might be associated with an element: "for each keyword in a list, get a picture and place it next to the keyword". These rules allow realizing loops. We call document elements that directly contain rules *control elements*. These are special complex document elements. Such elements also might contain more than one rule. *Layout elements* represent specific document structures, for instance tables, lists, or sequence and merge ordering instructions for included document elements.

The document composition model also defines a number of predefined *rule templates* that represent common complex situations and typical reactions and are configured by end users with respect to a specific mashup. Users can include these rule templates into mashups without having to define all rules from scratch. E.g., the "document element with table of content" rule template consists of a document element of type table of content and two rules: "show/include if mashup contains at least one chapter document element" (conditional document structure rule) and "update whenever a service in the mashup updates or mashup is changed" (interaction rule). Instead of a table of content any other table, for instance table of pictures or bibliography, can be used. The "reminder" rule template refers to all document services in the document element it is defined in. It contains the rule "at a certain date resend requests to all document services which did not deliver" (interaction rule). The date has to be configured by the user.

3. ARCHITECTURAL BLUEPRINT AND IMPLEMENTATION APPROACH

In order to enable open document collaboration via document service mashups, we have developed an architecture for a *document service infrastructure* and a *document collaboration environment* (see Figure 2). Our architecture reflects the major conceptual characteristics of document service mashups: the service infrastructure supports document service lifecycles and interaction and the collaboration environment facilitates interactive document mashup regulation and control.

For the service infrastructure, we have mapped the document service model onto RESTful services. Document services provide a minimal CRUD interface for document service messages. A fundamental document interaction protocol specifies the basic messages and messaging patterns of open document collaboration that can be either requests for, or lifecycle events from, document services. These messages are routed to any combination of mashups and services by means of a document service bus thus facilitating arbitrary interaction patterns that might be appropriate to

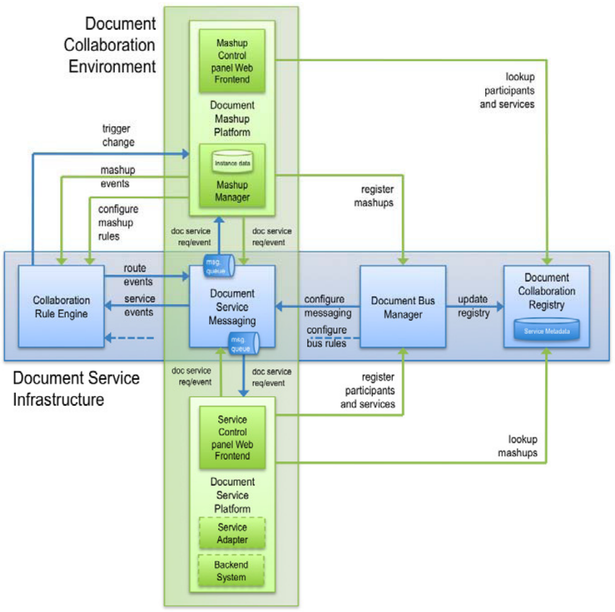

Figure 2. Overview of the MoSaiC Document Collaboration Architecture

enforce collaborative regulations. Routing decisions build on a set of interaction rules over the event stream going through the bus. A central rule engine provides respective capabilities of complex event processing (CEP). Furthermore, the document service bus maintains a knowledge base of ongoing collaboration processes and associated mashups as well as collaborators and their document service offers.

The collaboration environment reconciles between the interactive regulation of document collaboration processes by means of visual document mashup specification and the automated enforcement of these regulations by means of interactive as well as rule-driven document service interactions. Central to the environment is a document mashup platform. It interfaces with end users through a Web 2.0 application frontend that provides a runtime control panel for all aspects of a document mashup. In particular, a human participant in the role of a collaboration coordinator instruments the panel to supervise the collaboration process over its complete lifecycle. The mashup platform interfaces with the service infrastructure via a mashup manager component. During interactive mashup specification, the manager exchanges metadata on the mashup, its services and its participants with the document collaboration knowledge base. Then, its active behavior builds on the execution of mashup rules by the central rule engine. While the mashup manager feeds events on structural document changes to the rule engine, the rule engine triggers activities of the manager to alter the ongoing collaboration by changing the mashup. Predominantly, the manager requests collaborative activities by means of document service calls and receives events of ongoing activities in order to render their results as parts of the document mashup.

Another aspect of the collaboration environment concerns the integration of document services that might be either provided by human collaborators or backend systems. A Web application frontend allows collaborators to manage their collaborative activities, register respective document services with the infrastructure and view the document mashups that they are involved in. Depending on the type of collaborative activity, document services are either implemented as an editor for document fragments or by backend systems that are connected through adapters.

3.1 Document Service Infrastructure

In our collaboration architecture, an infrastructure layer mediates between document mashups and services. The layer consists of four components that interact with the mashup platform and the document services. The *document service messaging component* is the central communication channel that allows a decoupled message-oriented communication between the document services and the mashups they are used in. A *document collaboration registry* stores meta-information about document services and their providers. The mashup editor queries this registry to find those document services that are best suited for a given collaboration. New document service providers and mashups are registered at the *document bus manager*. It is connected to the messaging component and registry to configure them accordingly. The *collaboration rule engine* processes all event messages sent from document services and document mashups in order to evaluate the rules that control mashup execution.

The document service messaging component routes service requests between the mashup manager and document services. Document services are created and updated with a request to their uniform interface. It includes an XML message that contains meta information and additional data required for processing the request, e.g. human-readable advice. The services do not respond synchronously but propagate all state changes that result from a request as event messages. As the editing of document fragments may take time and occur in several steps, a single request may be followed by multiple events. To support this asynchronous communication model, the messaging component is based on a JMS message broker. The Java Messaging Service (JMS) is a standardized API that supports reliable messaging and point-to-point as well publish-subscribe communication. Each document service provider and each mashup have a message queue that stores request and event messages and forwards them when the service or mashup is available. Requests are sent to queues via a RESTful messaging API. It mirrors the interface of the document services so that the communication over the messaging component is transparent to the mashup manager and the services. As document services are addressed only by their unique ProviderID and ServiceID, their actual endpoint URI is hidden and mashup and services are decoupled. Event messages that contain the results of state changes of document services are sent to multiple recipients. They are processed in the collaboration rule engine and sent to all mashups and services that are interested in the results.

In our mashup infrastructure, we use a simple and user-friendly registry for service discovery. As the user decides which document services and which service provider he wants to use in a new mashup, advanced features like automated matching and selection are not required. Our document collaboration registry stores meta information about service provider, document service instances and document mashups and supports simple keyword queries on these attributes. If the user wants to create a new document service instance and add it to a mashup, he can search for a service provider that offers a certain document service type. Additionally, providers can register in a certain *collaboration context* so that their services are directly available in the corresponding mashup. The registry contains an informal textual description on how the expected content of the document service instance must be specified. Due to the uniform CRUD interface of document services, an operational interface description is not required. To allow reuse of existing content, the user can also query the registry for document service instances. If their creator did not restrict their reuse, they can be included in new mashups.

Document service providers that want to offer services in our mashup infrastructure must register with the document bus manager. They submit meta information about the document services they provide and may register service instances that already contain content that can be reused in other mashups. Providers may also register in one or more collaboration contexts so that their services are only available for use in corresponding mashups. The document bus manager stores the information in the registry and creates a queue for the provider in the messaging component. As some document service provider may not be online all the time, the manager allows suspending the forwarding of requests temporarily. The requests are stored in the queue of the provider and executed when it is available again.

The active behavior of the mashup is controlled by rules. Actions like calls to document services or changes of the mashup structure are triggered by conditions that are evaluated on the events emitted by services and mashups. As events from different documents services and mashups can be used in the rules, they are processed in the collaboration rule engine. To deal with the throughput of event messages in a conceptually centralized rule engine, we use the complex event processing engine Esper (http://esper.codehaus.org/). Esper allows querying event streams and detecting events patterns in them using the Esper Event Processing Language (EPL). EPL statements can contain event patterns that filter events based on their attributes, correlate multiple events using logical expressions and consider temporal constraints, e.g. on the ordering of the events.

We use EPL statements to formulate the conditions of the rules in our mashup platform. The following two patterns in EPL syntax demonstrate how typical collaboration rules can be expressed.

```
every ( Event (Source.ProviderID=‘Alice‘,
            Source.ServiceID=‘figure1‘,
            Type=’updated’)
        and Event (Source.ProviderID=‘Bob‘,
            Source.ServiceID=‘abstract‘,
            Type=’updated’) )
```

The pattern matches every time service “figure1” of provider “Alice” and service “abstract” of provider “Bob” are updated. It may be the condition of a rule that triggers a proofread service when both document services have changed.

```
not Event(ProviderID=‘Bob‘,
        ServiceID=‘abstract‘, Type=’updated’,
        state=’final’)
and timer:interval(24 hours)
```

This pattern matches if the provider does not emit a document fragment in final state within 24 hours. It can be used to remind the provider that a deadline has passed. To perform the corresponding actions, listeners are being attached to statements in the Esper engine. They are called when a pattern is matched and trigger actions in the mashup manager via a special interface.

Event messages can be sent to the Esper engine with different adapters. We use the JMS input adapter to connect it to the JMS message broker that is at the core of the Messaging Component. In that way, all event messages from the document services can be efficiently forwarded to the Esper engine when they pass the messaging component. In order to process the events that are emitted from the document collaboration environment (i.e. mashup events), the mashup manager is directly connected to the JMS message broker.

3.2 Document Collaboration Environment

The document collaboration environment supports user-based creation and administration of mashups. It is composed of two main parts, the document service platform and the document mashup platform.

The document service platform provides an infrastructure for managing provided services. Specific Web frontends allow creating and editing human-based document services, e.g. texts, using an editor but document services can be also implemented by backend systems that are connected through adapters. A *service control panel* Web frontend supports the registration of new providers and services. These interfaces access different functions of the document service infrastructure, e.g., if a new human-based service is created, it is deployed and registered at the bus using the document bus manager. A user interface for looking up mashups allows users to find mashups they participate in or register to mashups they want to participate in. Events caused by these update and registration actions are transmitted to the document service messaging component.

The document mashup platform contains two main components, the *mashup manager* as well as the *mashup control panel* Web frontend. The mashup manager holds the state of a mashup in cache and is responsible for reacting on events coming from the mashup control panel. Whenever a change for instance in the structure of a mashup is made by a user in the mashup control panel, the mashup manager is notified. It stores a new version of the mashup in its instance data repository. A mashup version contains the document structure built of document elements as well as the resource data of the associated service instances in their respective state. In addition, all rules are stored with a mashup. Whenever a rule is stored or updated in a mashup, it is also configured in the rule engine. Every mashup event coming from the mashup control panel is sent via JMS to the rule engine which can then check whether any follow-up actions are required.

The mashup manager in addition provides interfaces for actions which can be directly triggered by the rule engine. An example activity is the update of a service instance in a mashup. The rule engine calls the respective interface at the mashup manager which then stores a new version of the mashup including the updated service instance as well as updates the mashup control panel.

The mashup manager is also responsible for registering new mashups at the document bus manager which were designed by users using a visual editor of the mashup control panel.

The mashup control panel provides an intuitive Web-based GUI for collaboratively authoring mashups. A mashup can also be accessed as a document service in order to allow collaborators to view the collaborative document as a whole. In our document collaboration environment, we do not distinguish design time and runtime of a mashup; i.e., actions like changing the structure of a mashup or updating a service are performed immediately and do not need to be specified as a schema. In the Web frontend, the user can define and refine an overall document structure which is represented through the root document element of the mashup. Onto this structure, the user can drag and drop other document elements or document services from palettes. One palette lists existing document service instances and allows dragging and dropping them onto a document element in the mashup. A palette of providers and their offered services also allows to trigger the creation of new services: as soon as a new provider is dropped onto a document element, a create message is sent to the service provider which results in the creation of a service instance in "draft" state by the provider. The information about participants and services are directly retrieved from the document collaboration registry. The creation of rules is based on EPL; however we are developing a graphical user interface that will support the different types of rules, such that the user does not have to code. This includes a wizard for the interaction rules described in section 2.2 and a special view on the mashup, which allows defining document structure rules directly into the mashup.

4. EXAMPLE SCENARIO

We shall now motivate our vision for open document collaboration describing anecdotes of usage of an exemplary system, which could be based on our open document collaboration models.

Scientific publications involve collaboration of various researchers who provide different parts of the publication like pictures, texts, or references. Authors of a scientific publication need to discuss contents, structure the document, proof-read it, check for completeness, correctness and readability.

In our scenario, Alice starts creating a scientific publication mashup using a template that already provides some content structure like title, abstract and bibliography element. Alice uses the Web frontend and drags several section elements for introduction and main part from a palette to the mashup canvas. For each document element she provides a meaningful title and whenever Alice drops an element on the canvas, the mashup manager stores the document. In addition, for each change, mashup events are sent to the rule engine.

Having established the first structural draft, Alice defines behavioral rules. A wizard guides her through the process of creating the rule that all chapters need to be delivered three days before the conference deadline; i.e., at least update events for each of the chapters were emitted and they are in state final. First, a loop on all content document elements is defined. Second, an action is added to each element in the loop in order to send another request to the authors, who did not deliver. All rules are stored with the mashup. In addition, they are fed into the rule engine.

In the service instance palette Alice identifies a layout service, which formats mashups into ACM format and a service, which uploads a formatted document to the conference publication server. She defines a rule that the mashup is formatted by the layout service when all content elements are in final state. Afterwards a final proof-read by the first author is required and, in case the proof-read is ok, the document is published by the publication service to the conference publication system.

Having specified the rules, Alice uses the provider palette in order to find service providers and which services they offer. She finds

Bob, Carol and Ted who are in the research team and are planned to participate in authoring the publication. Alice drags the icon for Bob on the introduction document element; the association between service provider and service is immediately stored. Bob receives an e-mail that he has to write this paragraph and this task is added to his personal to-do list, which he can inspect after having logged in to the system.

Exactly three days before the deadline the rule engine triggers the "reminder" rule that was specified by Alice. It goes through the loop and for each document element that is not in state final, it triggers the action "send email to the provider". Since Bob did not provide any text for the introduction yet, he receives an e-mail, stating that he immediately has to deliver the introduction section for the publication. He logs into the system, writes the abstract and refines it adding a picture element into the introduction document element and marks its state as final.

The rule engine gets this event, and triggers the rule that as soon as all chapters are in state final, the mashup is sent to the layout service. After the layout service returned the formatted PDF file, the first author is requested to proof-read. After finishing the proof-read, the first author sends an event stating that the proof-read was "ok". This triggers the publishing action defined in the flow rule. The formatted document is send to the publishing service.

5. RELATED WORK

MoSaiC adopts concepts of *service mashups* for ad hoc composition of mostly *human-based software services* as parts of *modular documents* in the context of *open collaboration.*

Mashup technologies have recently gained broad attention from industry [2] and are increasingly addressed by academic research. The mashup paradigm, that emphasizes user-driven composition of situational software applications from Web-based content and services, is being adopted by a broad variety of approaches for general-purpose use like online developer platforms [9] or in the context of specific applications like e.g. collaborative story writing [12]. While a number of surveys have revealed general characteristics of existing mashup design and development approaches [6, 14], they also showed the need for new software technologies addressing specific requirements like collaborative development, user interface integration and ad hoc lifecycles. Among the specific challenges are high-level composition models that are appropriate for rapid end-user development like e.g. spreadsheets [15] and the integration of service presentation layers into a consistent user interface [5]. These challenges are also addressed by MoSaiC using a document-based approach.

The need for domain specific mashup technologies for document collaboration can be illustrated on the example of a general purpose mashup tool like the IBM Mashup Center (http://www-01.ibm.com/software/info/mashup-center/) that lets users compose widgets into mashups. Widgets reference services, e.g. feeds, from the Web or other sources like enterprise information systems. The definition of a widget holds information about which events the widget might expose. Based on this information, the mashup developer can add connection rules between widgets, which route data in case of an event to another widget. However, these events most often are caused by user clicks and not by updating any content. Events are exposed by widgets and not directly by services, the mashup or the platform, as required for collaborative applications. Likewise, it is not possible to specify complex rules involving several events or conditions. It is possible to collaborate on a mashup, but only the full mashup can be updated at a time and there is no role model or versioning. Also, the mashup still is an application and cannot be used as a document.

Currently there is a growing interest in technologies for *human-based software services* that allow integrating human intelligence task into automated information processing and harnessing the virtual workforce of Web users for highly scalable business processes [4]. An example is the the Human-provided Services framework [11] that aims to enable the engagement of humans in ad hoc and process-centric collaborations through representing human activities as services. This allows the seamless integration of human- and machine-based services. The approach focuses on interaction and collaboration, but not yet on composition of services into documents.

An early project on *modular documents* is XANADU [10], which started in the 1960s. It followed the vision of composing documents from pieces of other documents (transclusion), thereby building a networked bibliography of documents. The idea was to virtually copy contents, similar to links. This shows that the idea of composing documents from pieces is not new. However, XANADU did not focus on the collaborative evolution of documents. Furthermore, in our approach, the document elements are really copied into the mashup keeping the reference to the resource in order to request updates. They are not just referenced. More recently, Lublinsky discussed problems of incorporating documents into SOA implementations [7]. He proposes an approach to represent documents within a semantic data model that is used by business processes and services. The focus is however on rather static enterprise documents. Ad hoc modification of document structure and related processes is not addressed by this work. The document engineering approach by Glushko et al. [1] aims to integrate documents and business processes through analyzing and modeling the required documents and processes from a data, task, document, and business process perspective. These models capture common understanding of exchange information, timing, people, organization, or roles. However, this approach focuses on classical business processes and not on ad hoc and dynamic processes.

Regarding *open document collaboration* there exist various related Web 2.0 applications. Google Docs (http://docs.google.com/) allows collaborative creation of rich text documents. However, it is not possible to define any dependencies between document elements or react to events. Thus, lightweight ad hoc workflows cannot be specified and executed. Furthermore, Web services cannot be included into a document. Another related technology is Google Wave (http://wave.google.com/). A wave is a collaboration of participants based on XML documents consisting of wavelets. This is similar to the composition of document services to a mashup. Wavelets have a certain state defined by a sequence of operations. In addition waves might include automated robots that are comparable to our document transformation services since they provide functionality like translations. Wavelets can include any number of documents. However, Google Wave focuses more on communication than on collaborative evolution of an enterprise document. Also, there is no way to define interaction rules based on events and to re-use wavelets in other waves.

Finally, the Liquid Publications community investigates on new ways for the creation, dissemination, evaluation and maintenance of scientific publications by leveraging methods from agile software development and collaborative evolution of knowledge in Web 2.0 [8]. Similar to our approach, liquid publications are evolutionary, collaborative, reusable knowledge objects that can be composed. However, their focus is mainly on improving the notion of a scientific document but there is not yet a plan for a system, which could support collaboration on scientific publications. Thus, liquid publications are complementary to our research.

6. SUMMARY AND OUTLOOK

Open document collaboration poses substantial requirements on the flexibility of document structure and on the dynamicity of document contents. Document collaboration structure is highly situational and emerges in an ad hoc manner. Respective content has to reflect the evolving state of multiple participating collaborators and render its visual representation from various sources. Our approach of *document service mashups* provides a solution that meets both requirements. Document mashups support interactive specification of the structure and collaborative regulation of a shared document in a flexible declarative way on the fly. Document services facilitate the integration of dynamic content from various collaborators.

We have introduced a unique model of document service mashups that can be used as a basis to develop solutions for open document collaboration. Further, we have presented our technical architecture for such a solution that defines the main functional components and facilitates implementations on different platforms. We have also demonstrated the utilization of contemporary service-oriented infrastructure technologies including RESTful Web services, messaging patterns, complex event processing and rich Web 2.0 Internet applications for implementing the architecture and showing the feasibility of our approach.

Using the concepts of document service mashups, scenarios of open document collaboration can be soundly argued. Based on our architecture, the implementation of a complete demonstrator system is under active development. Prototypes exist for the document service infrastructure and document services. We are also assessing different Web application frontends like Wikis and custom RIAs. Experimental evaluation is ongoing. Apart from functional and qualitative tests that we have conducted for the existing prototypes, we will evaluate the approach in a case study experiment as soon as the complete demonstrator system is finished.

7. ACKNOWLEDGMENTS

We would like to thank Hewlett Packard for supporting the MoSaiC project as part of the Open Innovation Research Program.

8. REFERENCES

[1] GLUSHKO, R. J. AND MCGRATH, T. 2005. Document Engineering: analyzing and designing the semantics of Business Service Networks. In *Proceedings of the IEEE Eee05 international Workshop on Business Services Networks* (Hong Kong, March 29 - 29, 2005). ACM International Conference Proceeding Series, vol. 87. IEEE Press, Piscataway, NJ, 2-2.

[2] HOYER, V., AND FISCHER, M. Market overview of enterprise mashup tools. In *Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings* (2008), A. Bouguettaya, I. Krüger, and T. Margaria, Eds., vol. 5364 of *LNCS*, Springer, pp. 708–721.

[3] HUHNS, M. N., AND SINGH, M. P. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing 9*, 1 (2005), 75–81.

[4] KERN, R., ZIRPINS, C., AND AGARWAL, S. Managing quality of human-based eservices. In *ICSOC 2008 Workshops* (2009), G. Feuerlicht and W. Lamersdorf, Eds., vol. LNCS Vol. 5472, Springer, pp. 304–309.

[5] KONGDENFHA, W., BENATALLAH, B., VAYSSI`ERE, J., PAUL, R., AND CASATI, F. Rapid development of spreadsheet-based web mashups. In *WWW '09: Proceedings of the 18th international conference on World wide web* (Madrid, Spain, 2009), ACM, pp. 860, 851.

[6] KOSCHMIDER, A., TORRES, V., AND PELECHANO, V. Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web in conjunction with the 18th International World Wide Web Conference, Madrid.* (2009).

[7] LUBLINSKY, B. Unifying data, documents and processes. *Enterprise Architect 2*, 2 (2004).

[8] MARCHESE, M., GIUNCHIGLIA, F., AND CASATI, F. Liquid publications: Scientific publications meet the web. Techreport DIT-07-073, University of Trento, Department of Information Engineering and Computer Science, 2007.

[9] MAXIMILIEN, E., RANABAHU, A., AND GOMADAM, K. An online platform for web APIs and service mashups. *Internet Computing, IEEE 12*, 5 (2008), 32–43.

[10] NELSON, T. H. Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use. *ACM Computing Surveys 31*, 4 (1999).

[11] SCHALL, D., TRUONG, H. L., AND DUSTDAR, S. The human-provided services framework. In 10th IEEE Int. Conference on E-Commerce Technology (CEC 2008) / 5th IEEE Int. Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2008), July 21-14, 2008, Washington, DC, USA (2008), IEEE Comp. Society, pp. 149–156.

[12] SCHEIBLE, J., TUULOS, V., AND OJALA, T. Story mashup: design and evaluation of novel interactive storytelling game for mobile and web users. In *MUM '07: Proceedings of the 6th int. conference on Mobile and ubiquitous multimedia* (2007), ACM, pp. 148/139.

[13] SCHUSTER, N., ZIRPINS, C., TAI, S., BATTLE, S., AND HEUER, N. A service-oriented approach to document-centric situational collaboration processes. In *18th IEEE Int. Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises* (2009), IEEE Comp. Society, pp. pp.221–226.

[14] YU, J., BENATALLAH, B., CASATI, F., AND DANIEL, F. Understanding mashup development. *IEEE Internet Computing 12*, 5 (2008), 44–52.

[15] YU, J., BENATALLAH, B., CASATI, F., DANIEL, F., MATERA, M., AND SAINT-PAUL, R. Mixup: A development and runtime environment for integration at the presentation layer. In *Web Engineering*. 2007, pp. 479–484