Reliable Metering of Commercial Service Mashups

Christian Zirpins
Karlsruhe Institute of Technology,
Institute AIFB, Englerstr. 11,
76131 Karlsruhe, Germany
christian.zirpins@kit.edu

Elmar Jakobs
Karlsruhe Institute of Technology,
Institute AIFB, Englerstr. 11,
76131 Karlsruhe, Germany
elmar.jakobs@student.kit.edu

Volker Kuttruff
CAS Software AG
Wilhelm-Schickard-Str. 8-12
76131 Karlsruhe, Germany
volker.kuttruff@cas.de

ABSTRACT
On the Web, mashups have emerged as a promising approach to aggregate Web-based resources and services in an effortless way that allows for situational composition of software applications and in some cases even empowers end-users to do so. In this paper, we focus on commercial mashups as a means to implement customized service bundles in multi-level software service marketplaces and discuss problems that challenge reliable provision, aggregation and consumption of mashed-up services requiring payment. In particular, we propose a pragmatic method for service metering that considers the conflicting expectations from providers of mashed up services and consumers of service mashups in case of failures during REST-based service interactions over the Web. Our method builds on a relaxed version of the Try Confirm/Cancel protocol for transactional control. Our discussion builds on experiences gained in the COCKTAIL project that aims at a holistic platform for commercial service mashups. This platform provides the basis for a prototypical implementation of our metering approach.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: On-line Information Services – *Web-based services.*

General Terms
Management, Measurement, Reliability

Keywords
Commercial Service Mashups, Mashup Metering, RESTful Web Services, Try Confirm/Cancel

1. INTRODUCTION
Around five years ago, the term mashup was coined as an approach that allows for ad hoc composition of Web-based resources and services in order to create situational Web applications [3]. Since the early days of mashup technology, experimental systems such as *Yahoo Pipes* or *Microsoft Popfly* that were targeted mostly at end-users have evolved into professional tools like *IBM Mashup Center* or *SAP EMAP* that are targeted at professional enterprise contexts [2].

Meanwhile, mashups are also used to create customized commercial service bundles in the context of service marketplaces or SaaS platforms. For instance, the *CAS Pia* platform (http://www.cas-pia.de/) offers CRM solutions as SaaS to enterprise customers as well as PaaS functionality that not only enables third party providers of CRM services to join the platform but also provides mashup functionality to aggregate third party services into customized bundles. Because in this case mashups become commercial offers, they are subject to much stricter individual expectations (and legal obligations) than traditional mashup scenarios that mostly follow a best effort approach. This can lead to problems because neither the underlying Web-technologies (e.g. TCP/IP and HTTP) and generic data formats (e.g. RSS, JSON) nor the mashup development methodologies (mostly experimental trial-and-error-style development aiming at 'good enough' solutions) were originally designed for providing the level of reliability that is required by commercialization.

A specific problem in this context relates to *metering*: If a mashup fails during execution and some service components have already been invoked, the providers of these services rightfully expect payment but the consumer of the mashup equally rightful expects a useful result that can not be delivered and thus will be reluctant to pay. While this problem is classically solved by means of (business) transaction management [5], this would be inadequate for mashups aiming at lightweight composition models.

In our work, we have followed a design science research methodology to solve this problem in the context of a large-scale mashup commercialization platform that is being developed in the national German research project COCKTAIL. Our approach consists of a pragmatic *mashup utilization model* combined with a lightweight *mashup transaction protocol* on top of basic HTTP functionality. The approach suggests to structure mashups into units that provide self-contained benefit for clients and justify individual payment. These units are being checked for basic success-criteria (e.g. service reachability) prior to issuing invocations and becoming liable to pay. We have evaluated this strategy by implementing it as part of the COCKTAIL platform.

In this paper we aim to raise awareness for the challenges of mashup commercialization and introduce a pragmatic metering approach as a partial solution. Subsequently, Section 2 gives an overview of the COCKTAIL project and its approach towards mashup commercialization. Thereafter, Section 3 introduces our conceptual approach to service mashup metering followed by Section 4 describing its implementation. Finally, Section 5 discusses related work and Section 6 concludes.

2. MASHUP COMMERCIALIZATION IN THE COCKTAIL PROJECT
The COCKTAIL project (http://www.cocktail-projekt.de/) aims to develop a platform for commercial enterprise mashups. This encompasses technical challenges like authentication and single-

Mashups 2011, September 2011; Lugano, Switzerland.

sign-on, the management of mashups and external services together with their dependencies in a service repository and the composition of services/mashups to derive higher-level mashups.

Additionally, the COCKTAIL platform addresses business-related problems. While in the beginning of Web 2.0, available services as well as derived mashups were mostly free offerings (at least for personal use), enterprise related mashups might also rely (at least in parts) on commercial services that come at a cost. For example, an enterprise mashup that supports a marketing campaign could use the following three services:

1. A RESTful service invocation results in a list of possible recipients for a serial mail that serves as a basis for the selection of suitable address entries by the user. For example, this service can be provided by the CRM system used within a company.
2. A company that specializes on the translation of arbitrary documents translates a serial letter template written by the user. The process can either be an automated translation approach or a manual translation. Both translation types can be accessed using a RESTful service provided by the translation company.
3. The translated template and the selected recipients are sent to a service that results in a professional printing and cost-effective shipment of the letter.

These examples expose various different properties of commercial services. Most fundamental, services can be free of charge and return their results immediately (e.g. the CRM service). However, they might also come with costs. Provision of services might only require "virtual" resources like processing power (e.g. the automated translation service). Yet, service execution might also trigger long-running processes that consume physical resources (e.g. manual translation or printing and shipping of serial letters).

While easy and end-user friendly composition of services into mashups is still a vibrant research topic, aspects that arise from the commercialization of mashups are rarely considered. Practical experiences gathered during the development and integration of commercial REST-based Web services into an existing SaaS CRM system showed that a technical integration of services (i.e. development of a mashup) might be done by an experienced developer within reasonable costs. Interestingly, the complex and thus costly part was the preparation of the mashup and the referenced external services for *accounting*. Another complex issue was to cope with *exceptional situations* (i.e. runtime errors) during the execution of a mashup that uses services with costs.

Subsequently, an explicit goal of the forthcoming COCKTAIL platform is to provide means for simplified metering of service usage. Multiple independent service calls within one mashup invocation are embraced by a transaction. Although such a transaction does not support the ACID properties known from database transactions, it keeps together the different service calls that might occur over a certain period of time. The collected usage data is later used for accounting of mashup users and service providers. The platform supports different price models, e.g. flat rate or usage-based.

A lesson learned by the COCKTAIL partners (especially those from industry like CAS) is that accounting relies on robust metering of mashup/service usage as well as robust and adaptable mechanisms for error handling. We will discuss a pragmatic approach to address these challenges in the following sections.

3. METERING OF SERVICE MASHUPS

In order to account variable costs, the COCKTAIL platform uses a prepaid system for billing commercial mashup calls. This means that the user of a mashup application pays for the service in advance. Here, the problem occurs that the client pays for starting an application but does not get the expected results because of various possible failures. Another problem affects the mashup provider as he might deliver the output but does not get a compensation for it. The problem is of particular importance when the mashup provider has to call other commercial services during his execution because it might happen that he has to pay these services but does not get compensated resulting in a loss. This implies high risks for the mashup provider because in distributed systems (like mashups) and unsafe settings with autonomous service providers (like the Web), a lot of failures might occur.

A general solution approach for the above problem is based on *transactions*, as known from distributed database systems. Here, the atomicity guarantee provides that either the whole mashup application is executed or none of its services are invoked. The problem is that the HTTP protocol does not support transactional properties. Pardon and Pautasso introduce a possible way of realising atomic transactions over RESTful services [6]. Their approach builds upon the *Try-Cancel/Confirm* pattern (TCC). It proposes that resources should enter a reserved state at first. Only after a second confirmation-request, they reach their final state. If there is no confirmation during a specific time period or a cancelation is triggered then the resource reverts into its initial state.

In order to apply this idea to prepaid mashup metering, a mashup might be considered as a Web-accessible resource that consists of external sub-resources as well as an account holding the prepaid funds of a client. When being called by a mashup client the provider changes the local account (by deducing the mashup fee) only after completing a transaction comprising successful calls to all component services. However, such a transactional concept is problematic because mashup applications require loose coupling with respect to the ad hoc and often temporary aggregation of independent service components. Transactional control leads to a dependency between the mashup and the entire set of component services, which might be too coarse-grained. Furthermore, TCC requires specific REST APIs, where operations are separated into try/confirm parts, which might be too restrictive.

To solve this conflict, the idea of our approach is to apply transactional control only to *parts* of a mashup, where stronger dependencies are more acceptable than for the entire application. We refer to these parts as *mashup units*. More specifically, a mashup unit is a set of services providing a self-contained partial result that is usable and beneficial for the client as such and that the mashup provider can therefore bill the client for. As a result of introducing mashup units, transactional properties are only required for the individual units of a mashup and not for the mashup as a whole. This reduces the risk for the mashup provider even without transactional control, because in the case of a failure he would at most be liable to compensate the providers of a single mashup unit. Because of the reduced risk, the mechanisms of transactional control might be less strict. In particular, we suggest a pragmatic approach that mainly probes for the availability of the services within a mashup unit before committing the transaction.

To illustrate our approach, Figure 1 depicts an example for the transactional control of a mashup unit. In the diagram *Service-0* is the mashup application, which calls external services (*Service-2* and *Service-3*). The mashup unit comprises both external services.

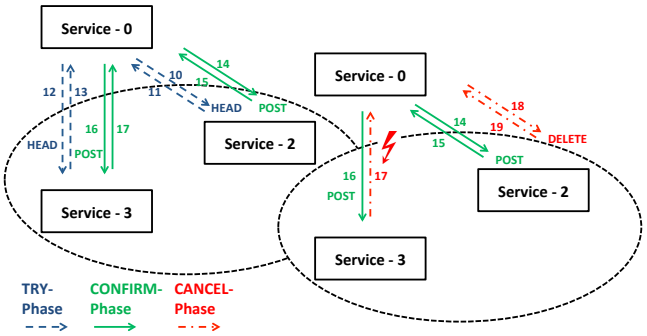

Figure 1. Transactional Execution of a Mashup Unit

Like in Pardon and Pautasso's transactional approach, there are different stages in the execution of a mashup unit. However, we modify the TCC protocol as follows:

In the trial stage, the mashup provider issues trial requests, which check the availability of the services composing the mashup unit. Therefore *Service-0* sends a HEAD request to both services. These requests do not interact with the service resources but just return the HTTP header of a potential GET request. If a reply contains the HTTP status code 200 OK, we assume that the service is available at least for a specific time period *t*. This corresponds to a degenerated form of trial in TCC: it does not guarantee success of subsequent service requests but allows to skip mashup units without liability to pay if any probe fails or times out. This trial just requires HTTP without any specific extension.

If the probes succeed, the mashup provider moves to the confirmation stage of the transaction and issues the actual service requests. Making these requests means that the mashup provider will be liable for paying the external service providers. If all service requests of the unit succeed, the mashup provider persists their aggregated results and deduces their combined service fees from the client account. If an error occurs in one of the service requests or if the time period *t* has passed without all requests having returned their results, the unit has failed. No changes will be made to the client account in this case and the mashup provider will try to rollback the mashup unit. Services that have not been called at this point will be skipped and thus do not need further attention. For all others, a HTTP request method has already been issued and therefore the provider tries to call a compensating request method to undo the action. While ultimately such compensating methods are specific to individual Web APIs, certain pairs of methods are natural counterparts, i.e. a POST of a resource is compensated by a DELETE of this resource and a PUT of a new resource is compensated by a PUT of the old resource. Compensating requests may have different consequences. For some cases it might be simple: If a running service can be aborted or its effective effort is low (e.g. when just returning database entries), it might offer free cancelation. For other cases, compensation might involve costs and requires payment (e.g. if physical actions have been carried out such as sending letters).

Compared to TCC, which requires specific operations separated into pairs of try/confirm parts that enable rollback by cancelation after the trial stage, the above method relies on a degenerated trial stage and rollback by means of compensation after the confirmation stage. This removes the requirement of a transaction-conformant API and is therefore potentially more suitable for mashups that need to cope with autonomous and heterogeneous services on the Web. This compromise is feasible if the average compensation costs are low. For the case of free cancellations the mashup provider can process a rollback without facing costs that he does not get paid for. Otherwise, he still gets paid for mashup units that have been already executed.

4. IMPLEMENTATION

To underpin the concept of lightweight transactional mashup units, we have implemented a prototype within the COCKTAIL project and evaluated it by means of a use case experiment.

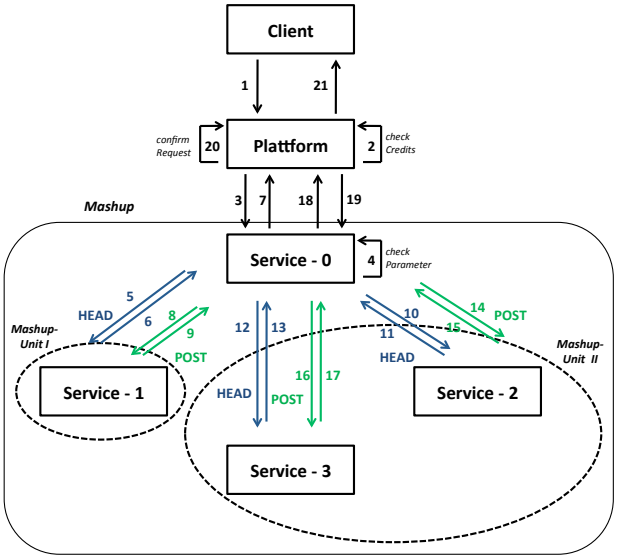

Figure 2. Interaction Protocol for Prepaid Mashup Invocation

More concrete, the concept of lightweight transactional mashup units manifests as an *interaction protocol for prepaid mashup invocation* that is illustrated in Figure 2. The most active component of this scenario is the COCKPIT platform itself, which does the metering and billing of commercial mashup calls. The platform acts as an intermediary between the client of the mashup and the mashup application itself. Also, the platform provides a service repository, which allows users to search for a mashup application and start it within their browser. After making the request, the platform checks the prepaid account of the requesting user and subsequently – if the balance is sufficient – triggers execution of the mashup application. The service-mashup then calls its external services as described before. After successful execution the platform returns the results to the user.

Assuming that services provide means for cancelation or compensation (no other behaviour is required from them), protocol implementation solely concerns the mashup platform. Therefore, the COCKTAIL platform has been extended by additional protocol logic for prepaid mashup invocation as shown in Figure 3.

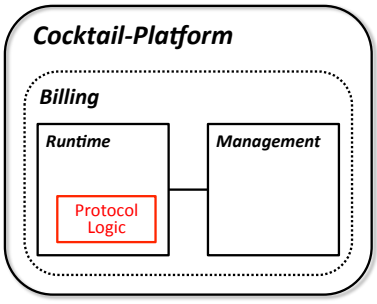

Figure 3. Cocktail-Platform

The platform contains a *billing component*, which provides the infrastructure for the prepaid provision of REST-based service-mashups. The billing component composes of the *runtime* and the

management part. The runtime provides functionality that is needed during mashup execution such as logging of mashup calls. The management part of the platform manages the prices associated with the different services. The implemented *protocol logic* is located in the runtime part but uses functionality from the management part, too.

For an experimental evaluation, a scenario has been implemented based on the mashup composition shown in Figure 2. It includes a mashup application and three external services. In this scenario, the mashup application provides a *credit scoring service*. Therefore *Service-1* returns datasets of specific clients from a CRM system. It also represents the first mashup unit. *Service-2* preselects customer records and *Service-3* calculates a credit scoring value for these remaining datasets. These two services make up the second mashup unit. The rationale is, that the user might not only be willing to pay for the final result but also for the datasets from the CRM-system, because he might be able to use them for another purpose. Here, the mashup provider benefits from the fact that he still gets a partial payment in case of a failure. For instance, he will get paid for mashup unit one if *Service-3* fails. Additionally, *Service-2* might offer a cancellation for free because filtering datasets does not involve high costs. So in this case the mashup provider can rollback the second mashup unit for free and has therefore no risks of losing money.

5. RELATED WORK

The work that we present in this paper focuses on pragmatic transactional control for metering lightweight aggregations of Web-based resources and services.

With respect to mashup commercialisation, a number of general aspects have been discussed in the research community. For instance, Zou et al. propose a framework and ontology to model accountability for mashup services [9]. Gangadharan et al. explore the actors and roles involved in a mashup ecosystem and analyse the intellectual rights associated with mashups [1]. Our work builds on such theoretical discussions and adds an application-oriented industry perspective.

Concerning transactional control, various approaches have been proposed for generic distributed systems including XA [4] and many others. Specific to Web services, Papazoglou has presented a study of requirements and characteristics of business transactions [5]. Techniques include *WS-BusinessActivity* that specifies a protocol for business transactions based on compensation or the compensation mechanism that is part of *BPEL*. However, such techniques are geared towards enterprise computing and mostly too heavyweight for mashup contexts.

Another line of related work concerns transactional control for RESTful Web services [7]. In the current discussion, REST-* approaches suggest to introduce heavyweight transactional protocols in the REST context [8] that are therefore generally inappropriate for our purpose. More suitable, Pardon et al. have proposed to apply TCC to RESTful Web services, which results in loosely coupled lightweight transaction control [6]. Our work is motivated by this approach and we leverage a relaxed version for our concept of more lightweight transactional mashup units.

6. SUMMARY AND OUTLOOK

In the light of current trends to commercialise virtually any IT resource as on-demand service over the Internet, mashup technologies offer a striking opportunity to unlock long-tail markets. Yet, commercial service mashups rely on a subtle balance between supporting ad hoc situational service aggregation (requiring loose coupling) and satisfying the rights and expectations of commercial service clients and providers (requiring transactional control).

As a possible solution, we have introduced lightweight transactional mashup units that leverage a relaxed form of TCC in order to make the metering of commercial mashups feasibly robust. Here, our notion of feasibility stems from practical industry experience: the respective mashup interaction protocol contents itself with optimistic indicative probes and relies on the relative ease and infrequence of cancelling/compensating the services of separate mashup units.

Supported by practical experience, our approach builds on certain assumptions such as the intrinsic value of partial mashup executions and the prevalence of service interaction patterns allowing for cancelation/compensation in conformance with our protocol. Other assumptions like the use of a prepaid billing system or a proprietary service repository are mainly owed to the existing COCKTAIL platform and we do not consider these as vital for our general concept. As regards the billing method, the prepaid case even addresses additional problems (concerning prepaid account management) that are not relevant in the normal case.

After having implemented our concept in the context of the COCKTAIL platform, we are in the process of gaining experience with it in practice. Initial experiments with different failure situations have shown the feasibility of our prototype and the approach as such. Ultimately however, a more extensive evaluation is needed, which is why we plan to further underpin the approach by means of a field study.

7. ACKNOWLEDGMENTS

This research has been supported by BMBF grant 01BS0821 of the German Federal Ministry of Education and Research.

8. REFERENCES

[1] Gangadharan, G., Weiss, M., D'Andrea, V., and Iannella, R. Exploring the intellectual rights in the Mashup ecosystem. *Proc. 6th Int. Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods*, Poznań University of Economics (2008), 13-22.

[2] Hoyer, V. and Fischer, M. Market overview of enterprise mashup tools. *Service-Oriented Computing–ICSOC 2008*, Springer (2008), 13-22.

[3] Merrill, D. Mashups: The new breed of Web app. *IBM developerWorks*, 2006. http://www.ibm.com/ developerworks/xml/library/x-mashups/index.html.

[4] Open Group. *Distributed TP: The XA Specification*. .

[5] Papazoglou, M.P. Web services and business transactions. *World Wide Web 6*, 1 (2003), 49–91.

[6] Pardon, G. and Pautasso, C. Towards Distributed Atomic Transactions over RESTful Services. In E. Wilde and C. Pautasso, eds., *REST: From Research to Practice*. Springer, 2011.

[7] Richardson, L. and Ruby, S. RESTful Web Services. 2007.

[8] Webber, J., Parastatidis, S., and Robinson, I. *REST in practice*. O'Reilly, 2010.

[9] Zou, J. and Pavlovski, C.J. Towards Accountable Enterprise Mashup Services. *IEEE International Conference on e-Business Engineering (ICEBE'07)*, IEEE (2007), 205-212.