A Distributed Transaction and Accounting Model for Digital Ecosystem Composed Services

Amir R. Razavi\textsuperscript{1}, Paul J. Malone\textsuperscript{2}, Sotiris Moschoyiannis\textsuperscript{1}, Brendan Jennings\textsuperscript{2} and Paul J. Krause\textsuperscript{1}

\textsuperscript{1}Department of Computing, School of Electronics and Physical Sciences, University of Surrey, Guildford, England, e-mail: (a.razavi,s.moschoyiannis,p.krause)@surrey.ac.uk
\textsuperscript{2}Telecommunications Software and Systems Group, Waterford Institute of Technology, Waterford, Ireland, e-mail: (p Malone,b.jennings)@tssg.org

Abstract—This paper addresses two known issues for dynamically composed services in digital ecosystems. The first issue is that of efficient distributed transaction management. The conventional view of transactions is unsuitable as the local autonomy of the participants is vital for the involvement of SMEs. The second issue is that of charging for such distributed transactions, where there will often be dynamically created services whose composition is not known in advance and might involve parts of different transactions. The paper provides solutions for both of these issues, which can be combined to provide for a unified approach to transaction management and accounting of dynamically composed services in digital ecosystems.

Index Terms—Distributed Services, Coordination, Transactions, Accounting, Digital Ecosystem.

I. INTRODUCTION

Many in the business and research communities are pursuing the vision of digital business ecosystems – distributed software environments through which organisations can seamlessly access customised, potentially disposable, services to aid them carry out a myriad of tasks. These services can be either pure software applications, or “real world” services represented by a software wrapper supporting automated transaction processing. Full realisation of this vision requires deployment of facilities for the dynamic discovery, composition, interoperation and execution monitoring of a potentially huge number of available services.

The European Commission funded Digital Business Ecosystem (DBE) FP6 integrated project [1] is building such an open source environment, through which businesses, in particular small to medium enterprises (SMEs), can interact within a pan-European ecosystem to provide access to arbitrary services that the DBE helps compose together to meet particular needs. The DBE is providing sophisticated structural and support software services that harness the scientific principles of self-organisation and self-optimisation to intelligently match organisations with dynamically composed services that are continually optimised to satisfy evolving business requirements.

However, the adoption of such a collaborative environment by SMEs is largely dependent on whether we are able to guarantee consistency and preserve local autonomy. Current frameworks [6, 7] tend to focus on providing consistency but, at the transaction level, this comes at the cost of violating the local autonomy of the participants.

Businesses utilising DBE services will need to be charged and billed in accordance with their service usage patterns. In any business environment charging processes are of crucial importance, thus the success of the DBE, or indeed any environment supporting dynamic service composition, will be contingent on the ability of providers to charge and collect fees for service usage. Given that there will be no a priori knowledge of the structure of, or business model associated with, dynamically composed services, the design and deployment of a sufficiently flexible accounting system is a challenging task.

In this paper we describe a model for coordinating distributed long running transactions. In our approach transactions are understood as pertaining to SOC for B2B interactions and thus a transaction has structure, comprising a number of sub-transactions, which involve the execution of different types of composed services. These are long-running and are coordinated locally, without breaking local autonomy.

We then describe how the accounting infrastructure of the DBE is being designed to cope with charging for dynamically composed services. In particular, we present a two-phase rating process for generating service related charges that allows service providers to straightforwardly dictate how their services are to be charged for when used in conjunction with other services they provide, or with services offered by other providers.

Both the transaction management and accounting solutions presented here are being made available through open source projects providing SMEs with affordable solutions to engage in eCommerce through digital ecosystems platforms.

The paper is structured as follows: after this introduction we provide a brief description of digital business ecosystems followed by overviews of both distributed transaction management and accounting for networked services focusing on the challenges specific to digital ecosystems. We then provide our solutions that address those challenges. Finally, we provide a summary of the work described in the paper and outline possible extensions.

The work presented here is funded partly through the EU Digital Business Ecosystems Integrated Project [1] and partly through the EU OPAALS network of excellence project.
II. THE DIGITAL BUSINESS ECOSYSTEM

The evolution of the Internet towards an integrated platform for service provision has been hampered by difficulties in efficiently translating new business requirements into deployed services. This has motivated significant research work on the specification of business modelling languages that can provide semantically enriched models of both business requirements and the services that meet them, with the aim of expediting the mapping between the two. Because of their semantic richness, these models can facilitate automated reasoning on the nature of service offerings, thereby supporting the construction, adaptation and evolution of service chains (or service compositions) that satisfy business requirements.

The DBE integrated project [1] focuses on the development of an open-source distributed environment (the DBE) that can support the spontaneous creation of applications through the composition of (not necessarily open-source) software services and components. In doing so the project is adopting the business modelling approach described above, but complementing it with the adoption of evolutionary algorithms inspired by biological processes, that provide bottom-up incremental improvement of business models through run-time feedback on service performance. The DBE is being targeted primarily towards SMEs, who will be able to concatenate their offered services within service chains formulated on a pan-European basis. By offering access to a large pool of service providers and consumers, and itself providing advanced recommendation systems and evolutionary algorithms, the DBE will support continued global optimisation of service chains, benefiting all actors, in particular SMEs [2].

From an architectural perspective the DBE can be viewed as consisting of a service factory environment and a service execution environment. Clients of the DBE will use the service factory environment to specify business models and generate associated software artefacts for subsequent implementation, composition and use. The service execution environment hosts implemented services, managing the process of registering, deploying, searching for, recommending, composing, retrieving and consuming services. In this paper we are concerned mainly with the service execution environment, specifically with the distributed transaction management, in a loosely-coupled manner, and the accounting infrastructure embedded and deployed therein.

**DBE Execution Environment**

Fig. 1 illustrates the main component types constituting the DBE service execution environment. Here we will provide a brief description of the functionality of these components; a fuller discussion of them can be found in [4].

The Service Composer (SC) component is responsible for constructing service compositions to meet specific requirements captured in the DBE service factory environment. The Recommender component provides the service composer with ranked lists of services that could fulfil specific functions within a service composition.

To rank services in this manner it uses historical information relating to service specification, use and performance, which is stored in the Knowledge Base repository. This information is organised as a “fitness landscape,” which assigns fitness levels to services on the basis of a range of criteria; the fitness landscape is also used as the main input into the evolutionary algorithms that optimise service compositions throughout their lifetimes. Once service compositions are constructed, their actual execution is coordinated by the Transaction Workflow Manager (TWFM). As individual services are consumed metering records detailing their utilisation patterns are generated and forwarded to the Accounting System, which is comprised of components for Mediation, Rating and Billing.

III. DISTRIBUTED TRANSACTION MANAGEMENT

The long-term nature of business transactions frames the concept of a transaction in Digital Business Ecosystems and makes defining a consistent transaction model even more challenging. The conventional view of a transaction [5] is based on the ACID (Atomicity, Consistency, Isolation, Durability) properties, which have been successfully applied to relational database management systems. However, in many new distributed applications such as CAD projects, e-commerce solutions and advanced simulators, these properties present unacceptable limitations.

It can be argued that the conventional view of a transaction cannot capture the primary requirements of DEs. From a business point of view, most usage scenarios in Digital Ecosystems involve long-term transactions and thus Atomicity is an unacceptable constraint. From a distributed transactions point of view, Isolation can lead to significant degradation of performance in the services offered (critical data is locked until a transaction completes) or to increased probability of deadlock (as services may be locked into composite transactions that do not terminate).

Transaction models for web services include WS-BusinessActivity [6] and BTP [7] among others. They are primarily concerned with consistency but in guaranteeing it the underlying coordination framework requires access to the internal build-up of the communicating parties. The Coordinator and Participant roles are tightly-coupled and if a fault occurs while the transaction is finalising (transition to Fault state while in Close state) it can only be dealt with at the participant’s platform. This not only implies that participating SMEs cannot have exclusive control of their local design but also that the underlying services are no longer
stateless. Such presumptions are against the primary requirements of SOA [9] and hinder the development of a Digital Ecosystem for SMEs.

In Section V we will introduce a transaction model that has been designed to overcome the above issues. However, before doing that we will review some material on accounting and charging. We will need this when we come to discuss the integration of our transaction model with a two-phase rating algorithm in Section VI.

IV. ACCOUNTING AND CHARGING FOR NETWORKED SERVICES

Accounting involves the collection and analysis of service and resource usage metrics for purposes such as billing, capacity and trend analysis, cost allocation and auditing. It requires that service consumption be measured, rated, and that resultant charging information be communicated between appropriate business entities. As shown in Fig. 2, accounting systems for networked services incorporate subsystems for metering, mediation, rating and billing.

Metering takes place within the network infrastructure; it is concerned with the accurate recording of service usage data and exposing collected usage records to the mediation subsystem. Mediation involves reliable collection of usage records from metering devices; correlation of records relating to the same service usage sessions; transformation of disparate metering record formats into a common format suited to the needs of the rating subsystem; and reliable transfer of processed records to the rating subsystem. Rating involves the application of models (charging schemes) for the mapping of usage data to monetary units based on various criteria. Each record received from the mediation subsystem is examined and the appropriate charging scheme is applied, resulting in the generation of a charge record. Finally, for post-paid customers, the Billing subsystem collates charge records for individual customers, who are invoiced on the basis of these charge records, and any additional subscription fees and discounts.

Accounting for services, including composite services, whose characteristics are known in advance is a mature area. For example, in the telecommunications domain there is wide deployment of complex mediation, rating and billing systems which support sophisticated usage- and content-based charging models for pre-paid and post-paid, private and corporate customers. In such systems the components involved in the accounting process must be manually pre-configured to account for specific services at the time those services are initially deployed. This will not be the case for composed services dynamically created and executed within a short time span – business logic for accounting must be automatically configured when service compositions are initially constructed, or subsequently modified.

Despite its importance, accounting for composed services has received little attention in the published literature. Bhushan et al. [3] propose a system architecture that supports the requirement for service providers to cooperate in the provision of composed services in a federated manner and share the generated revenue. However, they addresses only statically composed services – accounting components would still have to be pre-configured with the relevant accounting logic. Agarwal et al. [4] propose a method for metering and accounting for composite e-services that is not dependent on a-priori knowledge of the service composition. However, their approach supports only two specific service pricing models (flat rate per amount of resource used and flat rate per transaction). More significantly, the charge for an invocation of a composed service will always be the summation of the charges associated with standalone invocations of the constituent services; this will not always reflect the potentially complex business relationships between the service providers.

In Section VI below we will describe our two-phase rating approach, which addresses these issues. Before that we introduce the distributed transaction model.

V. DISTRIBUTED TRANSACTION MODEL

Our primary concern is with the support for long-term business transactions involving open communities of SMEs. Hence, service composition in this context is multidimensional and different forms of composition are needed to satisfy evolving business requirements. In contrast with the conventional view of transactions (data centric), web service compositions deal with at least three aspects: order, dependency, and alternative service execution [11].

In what follows we describe a transaction model that is expressive enough to capture different forms of service compositions and has the required structure to address the ordering of service invocations and the dependencies between them, as well as alternative execution scenarios.

In our approach, a multi-service transaction is represented by a tree structure (see Fig. 3). Each node of the tree is either a coordinator (can be understood as a composed service) or a basic service. The tree describes the coordination of the involved services and we may thus refer to the root of the tree as the master composed service. Basic services appear only as leaves of the tree. The resulting directed graph indicates the sub-transactions involved in a transaction and the corresponding hierarchical collection of basic and composed services. The coordinator nodes determine the ordering of execution and infer dependencies within the associated service hierarchy. We will describe how such issues, often collectively wrapped up in the term choreography [10], are addressed within our approach in the sequel.

First, we consider five different coordinator types, drawing upon [11], that allow for various forms of service composition to be expressed in our model.

1. Sequential coordinator: the services are invoked sequentially and the execution of a service is dependent on the previous one. This coordinator can handle sequential process-oriented service composition with provision for both
Sequential with Commit Dependency (SCD) and Sequential with Data Dependency (SDD).

2. **Parallel coordinator**: the services can be executed in parallel. This coordinator handles parallel process-oriented service composition covering Parallel with Commit Dependency (PCD), Parallel with Data Dependency (PDD) and Parallel without Dependency (PNP).

3. **Sequential Alternative** coordinator: the services will be attempted in succession until one produces the desired outcome, as specified by some criterion (e.g., cost, time).

4. **Parallel alternative** coordinator: alternative services are executed in parallel and once a service produces the desired outcome, the rest are aborted.

5. **Data-oriented** coordinator: this coordinator handles data-oriented service composition and specifically deals with released data items within a transaction (between its sub-transactions) or partial results released between different transactions.

6. **Delegation**: this coordinator allows the whole transaction or a sub-transaction to be delegated to another platform.

The first four coordinator types are rather self-explanatory. In long-lived transactions, partial results need to be shared between transactions before their termination (commitment). This is the purpose of the data-oriented coordinator. The delegation coordinator provides a means of overcoming traffic bottlenecks or low-bandwidth connections or (lack of) processing power. In this paper we are mostly interested in a charging scheme for distributed multi-service transactions and thus these coordinator types will not be covered in greater detail.

Consider the simple transaction tree depicted in Fig. 3 where we have adopted the notation of [8]. It comprises five basic services whose order of execution is determined by the three coordinators.

![Fig. 3 Multi-service transaction in a tree structure](image)

The “master composed service” is a sequential alternative coordinator defining two alternative execution scenarios: (i) service s1 followed by s2, and (ii) service s1 followed by s2, followed by s3, s4, and s5 in parallel. The second scenario of composed services will only be executed if the outcome of the first does not meet some preset condition.

The tree structure representation of a transaction allows us to exemplify the local coordination of the corresponding (compositions of) services that is required in performing the transaction in question.

To accommodate distributed long-running transactions that involve composed services, we need to relax the ACID properties, particularly Atomicity and Isolation without compromising Consistency. For this purpose, we need to consider some additional structure that guarantees the consistency of the transaction model within a highly dynamic and purely distributed environment of a Digital Ecosystem.

At the same time, we are considering SOC [9] as the primary computing paradigm for DBE. This entails that the model should defer from any tight-coupling between initiator and coordinator or between initiator and participant, as is the case with WS-BusinessActivity [6]. We therefore keep state information at the deployment level and abstain from interfering with service execution as such as we wish not to break the local autonomy of the realisation platform. Note that by ‘state’ here we refer to the activations within a transaction, in terms of its local coordinators and/or service invocations at any given point, and not to state as in the execution of the invoked services. The latter notion of state is related to the realisation level and is to be dealt with by the local platform on which these services actually run.

Next, we describe two graphs that capture the dependencies between sub-transactions (basic and/or composite services) of a single transaction or belonging to different transactions. Keeping track of such dependencies is essential if the underlying transaction model is to provide capabilities for charging for services, including composed services, and reverse action (in case some sub-transaction fails or is aborted). We also note that the two graphs are useful for applying deadlock control and for transparency during delegation. Such aspects are however beyond the scope of the present paper.

The **Internal Dependency Graph** (IDG) is a directed graph of arcs and nodes, which keeps logs of value dependencies within a transaction tree. Each node represents a coordinator or service (sub-transaction) and the direction of the arc between nodes indicates a dependency of one node on another.

In the transaction tree of Fig. 3, for example, s2 and s1 are children of a sequential coordinator and hence s2 is dependent on the results released by s1. This means that service s2 cannot be invoked before s1 has. It also has as a consequence that if s1 is aborted, then s2 must also be aborted. This dependency between s2 and s1 is shown in the IDG of Fig. 4(i).

The services s1, s3, and s4 are children of a parallel coordinator. This implies that when value dependencies exist if one of the services is aborted then the rest of the services must also be aborted. This is shown in the corresponding IDG given in Fig. 4(ii).

![Fig. 4 Internal Dependency Graph](image)

We have seen that in a distributed multi-service transaction, dependencies may exist not only between services of a transaction but also between services of different transactions. For instance, consider the case of (compensable) sub-transactions that release partial results in a conditional commit state [8]. This incurs a dependency between the corresponding service executions.
To capture such dependencies, we use another graph, called the External Dependency Graph (EDG). This graph keeps track of dependencies between (services or coordinators of) different transactions. The log structure it provides can also be used in recovery routines for running a compensating procedure, upon failure.

Fig. 5 shows (part of) the EDG for the transaction T1 (of Fig. 3) and transaction T2. It indicates that the service $s_1$ of T2 uses results released by service $s_2$ of T1, and is thus dependent on $s_2$.

The IDG and EDG provide a means of recording important system logs which can be stored locally, on the corresponding local coordinator, but their effect is both local, in terms of local faults, forward recovery and contingency plans, and global, in terms of abortion, restarting, recalculating, and alternative execution. Both graphs are used in the rating process of accounting, discussed in Section VI, for determining the final charges of composed services of a transaction (IDG) and across transactions (EDG).

VI. TWO PHASE RATING PROCESS

We return now to the rating algorithm. To meet the requirements outlined in §IV we propose that DBE rating engines employ a two phase rating process. In phase 1 of this process, services comprising a composed service are rated as if they are being executed as “standalone” services, leading to the generation of interim charges for each of these services. In phase 2, the interim charges are modified in accordance with provider specified rules specifying how charges are to be modified if that provider’s service(s) are used with other specified services in the context of a composed service. To facilitate the two phase process we utilise two-part charging schemes: part 1 dictates how charges are to be calculated when the service is invoked in isolation, and part 2 dictates any modifications to these charges if other specified services or service providers are present in the current transaction.

For the purpose of rating, we view composed services as hierarchical collections of atomic and composed services, in the same way that the transaction model is illustrated above in Fig. 3. We refer to the service at the top level of the hierarchy as the “master composed service”. Services at the bottom of the hierarchy must all be atomic (basic) services (from the rating engine’s perspective). All DBE services have associated with them a providerID, which uniquely identifies their provider within the DBE; they have a serviceID, which uniquely identifies the service amongst the set of services offered by their provider; and an instanceID which distinguishes between multiple instances of the same service type (for example, serving different geographical markets). Therefore, the tuple of (providerID, serviceID, instanceID) uniquely identifies a service instance across the entire DBE.

When a Transaction Workflow Manager invokes a master composed service it assigns a transactionID that uniquely identifies this invocation of the master composed service during the timeframe between initial invocation and the completion of all processing associated with that invocation. The Transaction Workflow Manager then provides all services comprising the master composed service with this transactionID. All Metering records relating to these services will contain this transactionID, as well as the providerID, serviceID, and instanceID. In addition, the Transaction Workflow Manager indicates to the rating engine that an invocation of the master composed service with the specified transactionID is commencing, and provides it with details of the associated service hierarchy. The presence of the transactionID in all metering records provides a means for the accounting system to identify a service’s execution context.

As the services are executed, metering records relating to them are generated and transferred to the rating engine; these records are rated using part 1 of the charging scheme associated with that service. This is phase 1 of the rating process; it results in the generation of interim charge records for the invocations of the associated service instances.

In the phase 2 of the process the rating engine modifies the interim charge records generated in phase 1 as dictated by the part 2s of the relevant charging schemes. These elements of the charging schemes capture relationships between services or between service providers and provide the rating engine with appropriate discounts (or tariffs) to be applied when such services have transaction dependency relationships as indicated by the IDGs and EDGs provided by the transaction workflow manager.

A worked Example

To show how this works in practice we will consider the simple scenario described above in Section V and depicted in Fig. 3. Let’s assume that each of the services has a very simple charging scheme comprising a flat rate per consumption as follows. The rating engine is designed to use a workbook approach to charging algorithms providing the user with a rich set of basic formulae to calculate arbitrarily complex charges considerably more complex than the simple flat rate charges shown here for brevity and clarity.

<table>
<thead>
<tr>
<th>Table 1 Providers, services and charges for scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>$P_1$</td>
</tr>
<tr>
<td>$P_2$</td>
</tr>
<tr>
<td>$P_3$</td>
</tr>
<tr>
<td>$P_4$</td>
</tr>
<tr>
<td>$P_5$</td>
</tr>
</tbody>
</table>
Provider $P_j$ has a relationship with provider $P_k$ whereby a discount of 10% is applied when their services are combined. Provider $P_j$ has a similar arrangement of 20% discount with $P_l$. Of the alternative scenarios, let’s suppose scenario 2 is executed. The output of $s_1$ followed by $s_2$ is unsuccessful (either service failed or did not produce the desired outcome) and this is aborted. Service $s_3$, $s_4$ and $s_5$ are then executed in parallel and are successful. The first phase of the charging algorithm in this case is a simple summation of $s_3$, $s_4$ and $s_5$.

<table>
<thead>
<tr>
<th>Service</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_3$</td>
<td>€0.20</td>
</tr>
<tr>
<td>$s_4$</td>
<td>€0.15</td>
</tr>
<tr>
<td>$s_5$</td>
<td>€0.20</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>€0.55</strong></td>
</tr>
</tbody>
</table>

We have seen that the parallel composition of the services $s_3$, $s_4$ and $s_5$, as indicated in the transaction tree of Fig. 3, incurs a dependency between them which is captured in the corresponding IDG, shown in Fig. 4(ii). The rating engine now takes into account the dependencies in the corresponding IDG, shown in Fig. 4(ii), and applies the respective charging schemes considering those dependencies. Since $s_3$, $s_4$ and $s_5$ are combined (in parallel), the agreed discounts between the corresponding providers apply. In this case the rating engine in the second phase applies a 10% discount to $s_3$ and a 20% discount to $s_5$.

<table>
<thead>
<tr>
<th>Service</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_3$</td>
<td>€0.18</td>
</tr>
<tr>
<td>$s_4$</td>
<td>€0.15</td>
</tr>
<tr>
<td>$s_5$</td>
<td>€0.16</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>€0.49</strong></td>
</tr>
</tbody>
</table>

This demonstrates how the transaction model described above can be combined with the two phase rating algorithm to provide for incentive based discounting (or tariffing) with dynamically composed services in digital ecosystems.

**VII. SUMMARY**

In this paper we have discussed issues for transaction management and accounting of automatically composed services in digital ecosystems. Solutions for overcoming these issues have been presented through models of transaction management and a two phase rating algorithm to enable business relationships to influence pricing for composed services whose structure is not known in advance. These solutions are being offered through open source projects enabling SMEs to take full advantage of these types of dynamically composed services both in terms of transaction management and monetary remuneration.

Within a Digital Business Ecosystem a number of long running transactions take place, involving various service compositions, and there is an increased likelihood that some subtransaction is aborted (due to a service not responding, platform failure, parallel alternative compositions, etc.). Preliminary analysis of our transaction model shows that it is possible to provide a compensating mechanism that warrants consistency. The log structures of the IDG and EDG capture the dependencies due to released results within a transaction and partial results between transactions, respectively. To ensure consistency at all times, such dependencies need to be taken into account so that all dependent subtransactions are also aborted. At the same time, it is also possible to address omitted results (non-dependent subtransactions that may have provided valuable results and need not be aborted) through a forward recovery routine, again based on the IDG and EDG.

Additionally, work is in progress on a formal behavioural description of the proposed transaction model, which can be subsequently used for the rating of the corresponding service compositions. By verifying the behaviour exhibited by the underlying service compositions, the testing and deployment of the transactions and their accounting scheme can be eased. The formal representation of a transaction also allows to identify the possible sequences, if any, of reverse actions for compensation and forward recovery.

**REFERENCES**


