

Specifying Flexible Charging Rules for Composable Services

Brendan Jennings, Lei Xu, Eamonn de Leastar
 TSSG, Waterford Institute of Technology, Ireland
 {lxu, bjennings, edeleastar}@tssg.org

Abstract

Where services are offered on a commercial basis, the manner in which charges for service usage are calculated is of key importance. Services typically have associated with them a charging scheme specifying the rules for charge calculation; schemes can range from the simple (such fixed charge per service invocation) to highly complex (where charges are calculated dynamically in order to influence customer demand and thereby optimise overall system performance). Typically, charging schemes are manually configured and verified prior to services being made available to customers – typically, a time consuming and expensive process. In environments where service compositions can be rapidly built and offered to customers, manual specification of a charging scheme for the service composition becomes untenable. In this paper we describe how charge modification rules associated with individual services can be used to flexibly govern how a service is charged for when it is used in the context of a composed service.

1. Introduction

Many computer science researchers are pursuing the vision of service-oriented architectures through which end-users can seamlessly access customized and potentially disposable services to aid them carry out a myriad of everyday tasks. Full realization of this vision requires deployment of facilities for the dynamic discovery, composition, interoperation and execution monitoring of pre-existing networked software services, possibly administered by different organizations, or originated by multiple developers. Significant research efforts are addressing the development of frameworks and process-oriented techniques for service composition, much of them focusing on specification of services in terms of formal process semantics. In particular, work on ontology languages by the semantic web community is demonstrating how specification of service capabilities

in a machine interpretable manner facilitates automated interaction between software entities.

Although increasingly powerful methodologies, languages and algorithms supporting both the construction, execution and adaptation of dynamically composed services have emerged, little attention has been paid to the supporting infrastructure necessary for their widespread deployment. Effectively managing systems in which the services provided can be dynamically composed is significantly more complex than that of managing systems in which the set of services offered to end-users is either known in advance, or is evolved in a carefully planned manner. Existing management systems target management of services and their non-functional properties on an individual basis; they do not consider the possibility that services can be collectively orchestrated in an almost arbitrary manner to fulfil changing requirements. It is unclear how these systems can be evolved to provide, for example, performance assurance, service-level security, accounting or fault detection for a constantly changing roster of composed services.

In this paper we consider one of the key issues regarding management of the non-functional properties of commercially-offered composed services – how we can effectively charge for their usage. Clearly, the commercial success of systems facilitating service composition is contingent on the ability of providers to charge and collect fees for service usage. Accounting and charging 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 accounting systems which support sophisticated usage- and content-based charging schemes. In such systems the components involved in the accounting process are manually configured to account for specific services at the time those services are initially deployed. This is a time-consuming and expensive process that increases the time-to-market of services, which has the undesired long-term effect of inhibiting the level of service innovation in the market. However, in environments

where services are frequently composed together the manual configuration approach is no longer tenable. Instead, accounting and charging logic should be *automatically* configured when service compositions are initially constructed, or subsequently modified.

In this paper we describe a framework for charging for composed service usage in which charging schemes associated with individual services can (1) be automatically combined when service compositions are constructed and (2) be deployed on the accounting system components (often termed rating engines) that calculate service usage charges. In particular, we describe how rules governing how service charges should be modified when a service is executed in the context of a composed service can be specified as part of a service's charging scheme. The goal is to reflect, in a flexible manner, the common business practice of price bundling of service offerings [1].

The paper is structured as follows: in section 2 we briefly review the state of the art in composed service accounting, whilst in section 3 we provide an overview of our framework for composed service charging. Section 4 describes our approach to specifying charge modification rules, whilst section 5 discusses the representations of modification rules using a Groovy-based Domain Specific Language (DSL) and provides a simple example. Finally, section 6 summarizes the paper and outlines areas for future work.

2. Related work

Processes for accounting and charging of services are realized by one or more systems incorporating functionality for *metering* and *mediation* (collection and filtering of service usage information), *rating* (application of models for mapping usage data to monetary charging units based on various criteria and creation of charge records reflecting individual usage sessions) and, for post-paid customers, *billing* (generation of invoices, collection of payments and customer account management) or, for pre-paid customers, *balance management* (decrementing appropriate charges from a user's current credit balance).

Given the critical importance of accounting and charging to any business, numerous standardization bodies have specified standards for charging systems, processes and protocols for a range of application domains. For example, in telecommunications, the 3GPP SA5 working group has specified standards governing the GPRS/UMTS mobile network entities involved in accounting, whilst the IETF Authentication, Authorization and Accounting (AAA) working group has specified protocols for the transfer of accounting data in IP networks. A detailed survey of

these and other efforts can be found in [2]. We note that the resulting standards universally assume that accounting and charging is done on an essentially per-service basis; they do not consider the complexities involved in charging for composed services in a manner that can flexibly reflect business agreements between multiple service providers.

Bhushan et al. [3] proposed a framework for Federated Accounting which addresses some aspects of charging for composed services; however, their work assumes that services are statically composed and that accounting logic for them is pre-configured. Agarwal et al. [4] propose a method for metering and accounting for composite e-services, which is not dependent on prior knowledge of the service composition. However, their approach supports only two specific service charging models (flat rate per amount of resource used and flat rate per transaction) and the charge a composed service invocation is always the summation of the charges associated with standalone invocations of the constituent services. Taylor et al. [5] propose an event-driven architecture for billing in service-oriented architectures, focusing in particular on multi-organizational federated information services. Their architecture incorporates a transaction manager within the customer-facing value-added service component, which coordinates the charging and billing process by interacting with the source information services (who calculate charges for their individual usage). Again, the total charge will be the summation of the charges calculated by the individual services, who will not be aware of the other services they are being used in conjunction with.

In previous work [6], we outlined an initial framework for charging for multi-vendor composed services in which charging schemes for standalone services can be applied together to calculate charges for sessions of a composed service, but without specific pre-configuration. Central to the framework is a two-phase rating process in which charges for individual services are first calculated as if those services were executed in isolation, and subsequently these interim charges are modified based on the presence of other services in the service composition. Whilst this framework offers a powerful solution it suffers from some drawbacks: firstly, the solution relies on rating engines being upgraded to implement the two-phase rating process, which would make deployment more difficult, and, secondly, charging schemes still must be specified in terms of a representation that can be directly applied by the rating engine (current rating engines typically employ low-level scripting languages, so their configuration is a time consuming and expensive process).

In further work [7], we outlined an amended framework in which an “Accounting Logic Generator (ALG)” component is responsible for automatically combining charging schemes relating to individual services into an overall charging scheme for the composed service. This scheme can then be translated, using model-driven development techniques into the representations required for deployment on the rating engine(s) that will calculate the charges for the individual services. In this paper we focus on a particular aspect of this framework: how the charge modification rules included in individual service charging schemes facilitate composed service charging schemes that reflect business relationships.

3. Framework overview

In this section we provide a brief overview of the framework for flexible composed services charging originally outlined in [7]. We describe the framework in terms of the nature of service compositions it supports and its architectural components, focussing in particular on the Accounting Logic Generator.

3.1 Composed service model

From the perspective of charging and billing, service providers can adopt a number of approaches in offering services:

- *Services are offered as “atomic” services:*
In this approach services appear as individual (non-composed) services for which metering records identifying the service type in question are generated, and for which specific charging schemes are available. Note that these services may actually be realized by the provider as compositions of other services, but for operational or commercial reasons the provider wishes to mask this;
- *Services are registered as atomic services and the provider itself controls generation of charge*

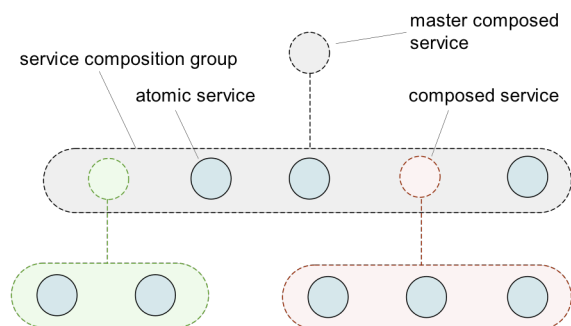


Figure 1. Composed service model.

information for the service:

In this approach services again appear as atomic services, even if they are actually realized as service compositions. However, the provider also takes responsibility for generating charge information for the service, either by maintaining its own rating engine, or by using a rating engine provided by a trusted third party. Thus, if the service is itself used as part of a service composition, its rating engine must be queried by the rating engine responsible for calculating the overall composed service usage charge;

- *Services are registered as composed services and the services comprising that composed service are identified:*

In this approach the structure of the offered composed service is visible within the execution environment. Metering records are generated by the (atomic) services comprising the composed service, and the accounting components identify these records as relating to an invocation of the composed service. The composed service may or may not have a specific charging scheme associated with it; if not, the rating engine must be able to calculate charges on the basis of the collection of charging schemes associated with its individual constituent services.

Given the above, we model composed services as hierarchical collections of atomic and composed services, as depicted in Figure 1. Each service is assumed to have a charging scheme associated with it that is comprised of two parts: part 1 indicates how charges are to be calculated if that services is used in isolation, whilst part 2 contains rules indicating how the charging scheme is to be modified if that service is composed together with other named services. Applying the part 2 modification rules allows the charging scheme for the composed service reflect the business relationships between service providers; for example, providers can respectively offer discounts, or impose penalties, when their services are used together, or in conjunction with services provided by business partners, or competitors.

3.1 Framework Components

The framework for flexible composed services charging is depicted in Figure 2. In the service environment a number of service providers offer their services, which can be used either on an individual basis or as part of service compositions. The focus of our work is on the Accounting Logic Generator component (described in section 3.2 below), but the

framework also requires the presence of the following components:

- *Service Composer (SC)*: responsible for constructing service compositions to meet specific requirements – this could be a manual, semi-automated or automated process;
- *Workflow Manager (WFM)*: responsible for coordinating the execution of individual service instances in the context of a composed service. The WFM could be a generic workflow engine, a separate hard-coded “service,” or a hybrid of the two;
- *Rating Engines (REs)*: apply charging schemes to map usage data to monetary units based on various criteria. Rating engines can be associated with individual services, individual service providers, or be provided as third party services themselves. The REs must support the WFM-to-RE and RE-to-RE protocols for composed service rating specified in [6];
- *Charging Scheme Specification Interface (CSSI)*: allows the specification of charging schemes for services. Currently, we assume that charging schemes are specified using a textual rather than a graphical interface.

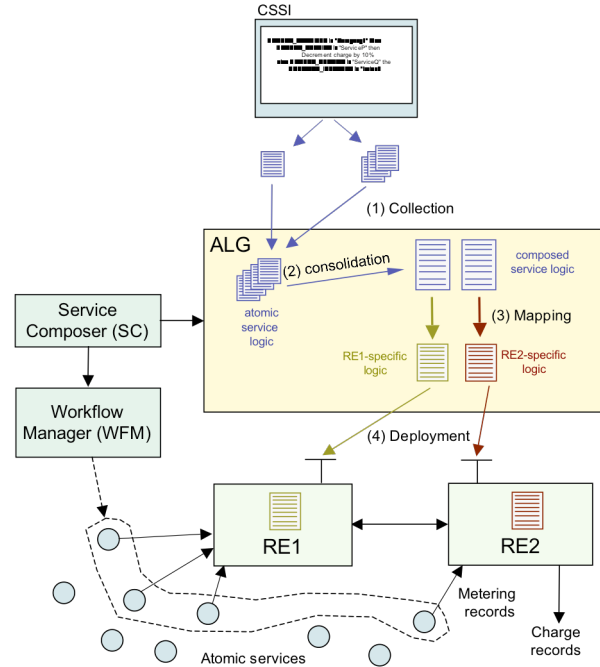


Figure 2. Framework for flexible charging of composed services.

3.3 Accounting Logic Generator

The central component in our framework is the Accounting Logic Generator (ALG), which is responsible for automatically generate charging logic to be deployed on a collection of rating engines to support a given composed service, whilst taking into account charge modifications introduced to incentivize or dis-incentivize particular service combinations. The ALG carries out the following functionality:

1. *Collection*
Upon creation of a composed service the SC will provide the ALG with details of the structure of that composed service. The ALG will then retrieve charging schemes for all the atomic services comprising the composed service from the location(s) indicated by the SC;
2. *Consolidation*
The ALG analyses the set of collected composed services and generates consolidated charging schemes, one for each of the target REs that the schemes will be deployed on. Note that providers may dictate that their services only be deployed on specific REs, hence there may be more than one RE involved in the calculation of the charge for a given composed service invocation;
3. *Mapping*
Once a consolidated charging scheme for

deployment on a given RE is generated it must be mapped to a representation that can be deployed directly to on that RE. Such representations may be in terms of a scripting language such as TCL, or in terms of table-driven representations such as spreadsheets;

4. *Deployment*
When RE-specific charging scheme representations are available the ALG will deploy them on that RE (this may be as simple as placing the scheme in a particular location in a file system). In addition to the information required to calculate charges, this scheme will contain information such as the unique identifier of the composed service.

A more complete description of the operation of the ALG, including an outline of the consolidation algorithm, is provided in [7]. Our focus in the remainder of this paper is on the nature of the charge modification rules (optionally) included in individual service charging schemes.

4. Charge modification rules requirements

In this section we describe the charge modification rules contained in part 2 of a service’s charging scheme. We assume that part 1 of a service’s charging scheme is the “default” and can always be used to calculate a single charge for a standalone invocation of that

service. Therefore, part 2 is constrained to specifying a modification of that charge. Given that our motivation for facilitating charge modifications for composed services is to give service providers the ability to incentivize service composers to use their services in conjunction with other services, we only introduce such modifications within a given composition group (such as the one identified in Figure 1). A service composer may choose to directly associate a charging scheme with a composed service it creates; this charging scheme may itself include charge modification rules to be applied if that composed service in case it is subsequently composed together with other services.

We assume that all service instances are uniquely identified by the tuple of (*providerID*, *serviceID*, *InstanceID*). The *providerID*, uniquely identifies, across the entire environment, the business entity offering a service. The *serviceID* uniquely identifies the service within the set of service types provided by the business entity with the service instance's *providerID*. The *instanceID* uniquely identifies an instance of a service type within the set of instances of services with the same *serviceID* and *providerID*. The latter reflects the fact that a service provider may maintain multiple instances of the same service type, and will potentially want to distinguish between these instances for charging purposes. Given these identifiers, it is possible to specify charge modification rules that effect modifications based on a service being composed with a given instance of another service, any instance of another service, or any instance of any service associated with a given provider. We believe this provides a sufficient degree of flexibility to satisfy most requirements.

We allow two types of charging scheme modification: charge decrement/increment by a specified percentage, or by a specified absolute amount (but optionally ensure a specified minimum charge). Whilst other possibilities exist (e.g. “decrement by a given percentage but cap the maximum absolute decrement at x currency units”), we believe this simple approach offers sufficient flexibility for most situations. To effect a decrement (or increment) by a fixed percentage the ALG can modify all the rates in part 1 of the service charging scheme accordingly before deploying it on a rating engine. To effect a decrement (or increment) of an absolute value the ALG can add a final step to the charge calculation process specified in the charging scheme part 1, before deploying it on a rating engine. Our framework also facilitates these modification types being applied more than once for a given service in a composition group (for example, if it is grouped with two partner services,

the charging scheme can be modified so that the sum of two absolute increments is applied to a charge).

Furthermore, modifications can be applied on the basis of individual customers or on groups of customers. Customers and customer groups are, in this context, considered to be customers of the provider offering the service in question and are thus uniquely identified by the tuples (*customerID*, *providerID*) and (*customerGroupID*, *providerID*), where *customerID* and *customerGroupID* respectively provide unique identification within the context of the customers of that provider. We assume that a mechanism exists to share such IDs between service providers and the WFMs that coordinate execution of composed services, since WFMs must provide these to REs for every service invocation basis. Additionally, the ALG must ensure that these modification rules are encoded in charging schemes deployed on the REs.

4.2 Error Handling

Our framework must also address how errors in the execution of a composed service potentially impact on the charging process. Clearly, during any service execution there is potential for both critical errors that force immediate halting of the service execution, and less serious errors that can be recovered from. Where a service halts due to error in the middle of a larger composed service execution a number of other services may have already completed fully, or be partially completed, and a number of other services may not have yet been invoked. In such a scenario Taylor et al. [5] argue that the customer should not have to pay, as the overall service they are using has not completed successfully. However, we believe this approach is too restrictive, as in many situations today customers must pay for partially delivered services, depending on the terms and conditions of their contract with a service provider.

To maximise flexibility our framework can provide for zero, partial, or complete payment for successfully completed services in the context of composed service invocations where one or more other services fail. This is achieved as follows: part 2 of service charging schemes may contain rules indicating what level of charge is to be levied in failure conditions. A service composer should take account of these rules when using the service as part of a composed service, and tailor customer contracts and/or the charge modification rules of the composed service accordingly. It should be noted that we assume that the charging rules specified in charging schemes are error free – our scope relates only to the correct execution of these rules, whose efficacy should be tested elsewhere.

```

def demo_service=new Service()
def builder=new DemoBuilder()
void scheme(){
    builder.foo(){
        SERVICEID_is "foo"
        INSTANCEID_is "France"
        PROVIDERID_is "Jennings"
        //PART_1:
        CHARGE_is 10
        //PART_2:
        If (OTHERPROVIDER.is("Xu_Inc")){
            If (OTHERSERVICE.is("Hat")){
                DECREMENT_CHARGE_PRESENTAGES_BY 10
            }
            Else_If (OTHERSERVICE.is("Bar")){
                If (OTHERINSTANCE.is ("Ireland")){
                    If (OTHERCUSTOMERGROUP.is ("Beckett")){
                        DECREMENT_CHARGE_PRESENTAGES_BY 12
                    }
                    Else_If (OTHERCUSTOMERGROUP.is ("Joyce")){
                        DECREMENT_CHARGE_PRESENTAGES_BY 5
                    }
                    Else {
                        DECREMENT_CHARGE_PRESENTAGES_BY 3
                    }
                }
                Else {
                    DECREMENT_CHARGE_PRESENTAGES_BY 10
                }
            }
        }
        Else{
            DECREMENT_CHARGE_PRESENTAGES_BY 5
        }
    }
    If (OTHERPROVIDER.is("deLeastar_Inc")){
        If (CHARGE>=11){
            DECREMENT_CHARGE_BY 3
        }
    }
    If (CONDITIONS){
        DECREMENT_CHARGE_PRESENTAGES_BY 70
    }
}
}

```

Figure 3. Charging scheme for service “foo”.

5. Charge modification DSL examples

In this section we provide a simple example of the use of charge modification rules to govern the charging of a composed service. We first discuss the use of a Groovy-based DSL to represent the modification rules and then provide a simple example of modification rules associated with two services and the charging scheme that results from their consolidation by the ALG.

5.1 DSL implementation

In implementing a DSL, Fowler [8] has identified two mechanisms: external and internal. An external DSL will require a grammar, a parser coupled with an interpreter or compiler to generate a complete round

```

def demo_service=new Service()
def builder=new DemoBuilder()
void scheme(){
    builder.bar(){
        SERVICEID_is "bar"
        INSTANCEID_is "France"
        PROVIDERID_is "Xu_Inc"
        //PART_1:
        CHARGE_is 5
        //PART_2:
        If (OTHERPROVIDER.is("Jennings_Inc")) {
            If (OTHERSERVICE.is("bar")) {
                DECREMENT_CHARGE_BY 0.8
            }
            Else{
                DECREMENT_CHARGE_BY 0.5
            }
        }
        If (OTHERPROVIDER.is ("deLeastar_Inc")){
            INCREMENT_CHARGE_BY 3
        }
    }
}

```

Figure 4. Charging scheme for service “bar”.

trip from program source to execution. This has long been at the heart of early approaches to tool development, for example in Unix (lex and yacc [9] being venerable early contributions). To this day many Unix distributions support diverse “little languages” for specific domains (text processing most prominent). An internal DSL however can be considered a more lightweight approach, relying in the inherent flexibility of the “host” language. Essentially the DSL is defined within a given programming language – effectively becoming a dialect targeted at the selected domain.

In this work we have selected the programming language Groovy for our initial experiments. Groovy [10] is a dynamically typed programming language, inspired by the Ruby programming language, but compatible with Java. Groovy programs execute on a standard Java Virtual Machine and can be developed within most Java IDEs. Dynamic should not be confused with weak – Groovy is in fact a strongly typed language – this has important security and safety implications in the charging domain. Significantly, Groovy’s dynamic nature enables a flexible and malleable syntax suitable for internal DSL construction. Groovy also has a number of syntactic innovations that facilitate small but significant tweaks to program structure that can yield surprising improvement in readability. Specifically, Groovy has flexibility in the following areas:

- semicolons are optional;
- variables can be introduced without explicitly type information;

```

package dsl.example
import dsl.service.Service

class foobar extends Service { void chargingScheme()
{
    def foo;def bar
    SERVICEID = "FooBar"
    INSTANCEID = ""
    PROVIDERID = "MBV_Inc"

    PART_1 =
    {
        foo.CHARGE = 9
        bar.CHARGE = 5
        CHARGE=foo.CHARGE + bar.CHARGE -0.8
    }

    PART_2 =
    {
        if (OTHERPROVIDER.is("deLeastar_Inc"))
        {
            DECREMENT_CHARGE_PRESENTAGES 5
        }
    }
}
}

```

Figure 5. Charging scheme for “foobar”.

- methods can be defined independent of classes;
 - lists and maps are built into the language;
 - ranges are first class objects.
- In addition, the language has more significant innovations that can have powerful impact on DSL design:
- Closures: a type of delegate model enabling a function plus associated scope to be incorporated into a single reference;
 - Meta Object Protocol (MOP): a mechanism for dynamically accessing the type of an object at runtime, introducing new fields or method for instance;
 - Builders: a design pattern that is given significant leverage by closures and MOP, producing a startlingly effective syntax for construction hierarchal data and behavioral models.

Taken together these features facilitate DSL creation in a familiar industry standard environment. The most successful example of a DSL in Groovy to date is the Grails Framework [11]. This is a DSL that encapsulates a high level model of the frameworks and infrastructure require by a web application. A range of other DSLs have been constructed in fields such as GUI Development, database access, scripting etc. (examples are described at [12]).

5.2 Example charge modification rules

We show implementations in our Groovy-based DSL of charging schemes associated with two services

and the charging scheme generated as a result of their consolidation by the ALG. We assume the simplest possible charging scheme part 1 – that of applying a fixed rate per invocation of the service. Figures 3 and 4 show the DSL implementations charging schemes for the two services, “foo” and “bar”.

Service “foo” from provider “Jennings_Inc” has a default (part 1) charge of 10 currency units every time it is invoked. When it is composed with the “France” instance of service “Bar” from provider “Xu_Inc” its part 2 indicates that a decrement of 10% is required (for all “Jennings_Inc” customers). Similarly, from the point of view of service “bar” from provider “Xu_Inc”, when it is composed together with any service from provider “Jennings_Inc” a decrement of 0.8 currency units is specified.

When the ALG receives this charging scheme information from the SC it consolidates them into the charging scheme for service “foobar” as shown in Figure 5. In this scheme the rate change required for service “foo” is applied as a direct rate change for this service. On the other hand, for service “bar” the rate does not change; instead, the absolute decrement of 0.8 currency units is applied in the last step of the part 1 calculation. In Figure 5 we also show a part 2 for service “foobar”. These part 2 rules would have been directly supplied to the ALG by the SC together with the individual service charging schemes. These modification rules are independent of the automatically generated part 1 of the “foobar” scheme and are only used if service “foobar” is itself subsequently used as part of a service composition.

6. Summary and Future Work

In this paper we described a framework facilitating automated generation and deployment of charging schemes for composed services that embody charge modification rules reflecting the business relationships between different service providers and between individual service providers and their customers. In particular, the paper focussed on the use of rules for charge modification as a means of reflecting business relationships between service providers (and customers) with the goal of incentivizing or disincentivizing the use of services with other providers’ services in the context of a composed service. This approach will be applicable regardless of the means of service composition, but is particularly suited to environments supporting dynamic service composition, in which charging rules for newly generated compositions would themselves have to be generated and applied on-the-fly.

We believe that flexible approaches of this nature will be required to ensure that innovation in the service market is not hampered by the need for costly and time-consuming configuration of charging logic for composed services. For future work we will extend the framework to cope with more complex service composition models, such as that outlined in [13]. The framework will then be realised and evaluated in the context of an IP Multimedia Subsystem test-bed supporting SIP-based communications services.

An area for further investigation is provision of a means of providing “advice-of-charge” to customers in advance of the invocation of a given composed service. Given the flexibility of the proposed approach it will not always be straightforward to identify what the actual charge for a service invocation will be, given that numerous discounts or penalties could be applied. We also intend to investigate the degree to which calculation of likely charges can play a role in the service selection process for service composition. For example a service composer may want to identify the cheapest collection of services that together perform a given task, which may not be a straightforward task given the complex business arrangements that may exist between different service providers.

7. References

- [1] H. Simon, and G. Wuebker, “Bundling – A Powerful Method to Better Exploit Profit Potential,” in R. Fuerderer, A. Herrmann and G. Wuebher, eds., *Optimal Bundling (Marketing Strategies for Improving Economic Performance)*, Springer, Heidelberg, 1999, pp. 7-30.
- [2] M. Koutsopoulou, A. Kaloxylos, A. Alonistioti, L. Merakos, and K. Kawamura, “Charging, accounting and billing management schemes in mobile telecommunication networks and the Internet,” *IEEE Communications Surveys*, vol. 6, no. 1, 2004, Online: <http://www.comsoc.org/pubs/surveys> [accessed: March 12, 2008]
- [3] B. Bhushan, M. Tschichholz, E. Leray, and W. Donnelly, “Federated Accounting: Service Charging and Billing in a Business to Business Environment,” in *Proc. IFIP/IEEE Int’l Symp. on Integrated Network Management (IM 2001)*, IEEE, 2001, pp. 107-121.
- [4] M. Agarwal, N. Karnik, and A. Kumar, “Metering and accounting for composite e-Services,” in *Proc. 1st IEEE Int’l Conf. on E-Commerce (CEC 2003)*, IEEE, 2003, pp. 35-39.
- [5] K. Taylor, T. Austin, and M. Cameron, “Charging for information services in Service-Oriented Architectures,” in *Proc. IEEE Int’l Workshop on Business Services Networks (BSN 2005)*, IEEE, 2005, pp. 16-23.
- [6] B. Jennings, and P. Malone, “Flexible Charging for Multi-provider Composed Services using a Federated, Two-Phase Rating Process,” in *Proc. 2006 IEEE/IFIP Network Operations & Management Symp. (NOMS 2006)*, IEEE, 2006, pp. 13-23.
- [7] L. Xu, and B. Jennings, “Automating the Generation, Deployment and Application of Charging Schemes for Composed IMS Services,” in *Proc. 10th IFIP/IEEE Symp. on Integrated Management (IM 2007)*, IEEE, 2007, pp. 856-859.
- [8] M. Fowler, “Domain Specific Languages,” [online] available (25/4/2008): <http://martinfowler.com/dslwip/>.
- [9] J. Levine, T. Mason, and D. Brown, “lex & yacc (2nd edition)”, , O’Reilly, 1992.
- [10] D. Koenig, A. Glover, P. King, G. Laforge, J. Skeet, and J. Gosling, “Groovy in Action”, Manning Publications, 2007.
- [11] G. Rocher, “The Definitive Guide to Grails”, Apress, 2006.
- [12] [Online]. Available (25/4/2008): <http://groovy.codehaus.org/>.
- [13] A. R. Razavi, P. Malone, S. Moschoyiannis, B. Jennings, and P. J. Krause, “A Distributed Transaction and Accounting Model for Digital Ecosystem Composed Systems”, in *Proc. Inaugural IEEE/IES Conf. on Digital EcoSystems and Technologies (DEST 2007)*, IEEE, 2007, pp. 63-66.