FinGrid Accounting and Billing

Houssam Haitof[1], Hans-Dieter Wehle[2], Michael Gerndt[1]

[1]Institut für Informatik
Technische Universtität München
Boltzmannstr. 3
85748 Garching, Germany
{haitof, gerndt}@in.tum.de

[2]IBM Deutschland Research
& Development GmbH
Schönaicher Str. 220
71032 Böblingen, Germany
hdwehle@de.ibm.com

Abstract: For a commercial entity to entrust the Grid for its business operations either as a consumer or a provider of resources, mechanisms that would guarantee its interests need to be implemented. Especially resource usage tracking and billing. In this paper, we present our experience with designing and building an autonomic accounting and billing system for the Financial Grid (FinGrid). We used a service oriented architecture for FinGrid, we relied on open standards and recommendations for our accounting system and on knowledge representation and reasoning to model our billing infrastructure.

1 Introduction

Grids consist of a virtual platform for computation and data management using a heterogeneous cluster of computer resources [BHF03]. It enables users and applications seamless access to vast IT capabilities. In his reference paper [Fos02], Foster provides a three-points checklist that a system has to fulfill before it could be identified as a Grid. The first is that [the Grid] *coordinates resources that are not subject to centralized control*. Meaning that not only the Grid resources are geographically distributed but that those resources belong to different administrative domains, i.e. different institutions or departments. Issues like security, usage policies, accounting and billing arise. Grids are often based on the "best-effort" principal that does not guarantee a sophisticated level of quality assurance. This may be quite satisfactory for an academic usage, however, Grids are more and more used and adapted for a commercial usage. And for a commercial entity to entrust the Grid for its business operations either as a consumer or a provider of resources, mechanisms that would guarantee its interests need to be implemented. In the frame of the FinGrid project, we built a general purpose, standard-compliant, Grid accounting system that tracks resources usage for tasks, like the management, SLA enforcement or billing. We also built a rule-based billing system, seamlessly integrated with the accounting module. The billing system allows to generate bills from the the resource usage according to a set of billing policies. All the components of FinGrid are composed within a SOA model.

The Financial Service Grid (FinGrid) is a project funded by the German Federal Ministry of Education and Research, to develop a Grid architecture to virtualize services and processes in the financial sector and to build banking Grid services based on an accounting

and pricing infrastructure through the development of several prototypes. In this context, we pursue research on the necessary components for a financial Grid to better model an industrialization and pricing scheme. We draw the architecture and implemented the resulting accounting and billing services. Sections 2 is about FinGrid SOA model, sections 3 and 4 will present FinGrid accounting and FinGrid billing respectively. Section 5 introduces our system architecture. In section 6 we will talk about the related work in the field and in section 7 we present our future work.

2 FinGrid SOA Model

Service Oriented Architecture (SOA) guidelines and web services technologies can be used to construct a solution for a flexible model for Grid management that would tackle part of the Grid management complexity challenges. According to OASIS [MLM+06], SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. Some of the main drivers for SOA are to *facilitate the manageable growth of large-scale enterprise systems and to facilitate Internet-scale provisioning and use of services*. It revolves around the concept that needs or requirements of one party are met by the capabilities of another. The parties at either ends can be person or a software agent. Applying SOA principles to the Grid seems to be a natural process. The Grid is composed of a set of distributed resources under the control of different administrative domains, and SOA is a model for organizing such system. In fact, the standardization efforts for the Grid are channeled towards the adoption of web services technologies that lay the necessary infrastructure and building blocks for a Service Oriented Grid Architecture.

In a SOA, the central mechanisms for coupling needs and capabilities are services, defined by the capability of performing work for another, specifying the work offered for another and the offer to perform work for another. The first step to meet SOA objectives is done through decomposition or factoring of complex systems into smaller chunks for more convenient design, implementation and maintenance. Those smaller chunks are what services are supposed to be: small independent components easier to manage and control. The SOA model does not preconize the specific use of web services, however they constitute the used de-facto standard.

FinGrid accounting and billing contains several small services composed together to constitute an autonomic management layer whose function is to automatically collect usage information from the different Grid nodes using collector agents and produce usage bills. Every service is independent and can be recomposed in a different context. The main FinGrid services are the Grid accounting, Grid billing, Grid rules Manager, job submission and collector agents. In this paper we present the accounting and billing architecture and thus concentrate on the accounting, billing and collectors services.

3 FinGrid Accounting

The purpose of this component is to provide an interface for collecting (upload) and accessing (download) accounting information of the Grid resources consumption and provision. Accounting is the collection of information and data on the usage of Grid resources resulting in a report of the resource consumption and/or provision. The resulting report is generally used for capacity planning, trend analysis, auditing, billing and/or cost allocation. Gathered accounting information is in the form of an XML document complying with the Usage Record (UR) Recommendation[MLMJM07] proposed by the Open Grid Forum (OGF).

3.1 Usage Record

The UR recommendation specifies a common format for representing resource consumption data. It contains accounting and usage information gathered at the local Grid sites. The usage metrics are divided into three categories: base properties, differentiated properties, and extensions. The base properties define the most common metrics necessary for proper accounting such as user and job identification. The differentiated properties are job-related measurement metrics that every Grid site can accommodate to its particular needs. The last category is the extensions, which are a set of metrics specific to the Grid site and jobs that can be added to the UR specification. The set of required items to be accounted is, of course, site and situation dependent. The listing below shows a sample UR document generated in our testing environment.

```
 1 <urwg:UsageRecord>
 2 <urwg:RecordIdentity urwg:createTime="2007-11-20T10:
    59:30Z" urwg:recordId="2007-11-20T10:59:28Zfingrid.
    boeblingen.de.ibm.com@griduser"/>
 3 <urwg:JobIdentity>
 4 <urwg:GlobalJobId>https://9.152.4.12:8443/wsrf/
    services/ManagedExecutableJobService?4499c150-974f-
    11dc-ab80-852cb8646fdc</urwg:GlobalJobId>
 5 <urwg:LocalJobId>fingrid.boeblingen.de.ibm.com#119555
    2763#147.0</urwg:LocalJobId>
 6 </urwg:JobIdentity>
 7 <urwg:UserIdentity>
 8 <urwg:LocalUserId>griduser</urwg:LocalUserId>
 9 <ds:KeyInfo>
10 <ds:KeyName>/O=Grid/OU=GlobusTest/OU=simpleCA-fingrid
    .boeblingen.de.ibm.com/OU=boeblingen.de.ibm.com/
    CN=Grid User</ds:KeyName>
11 </ds:KeyInfo>
12 </urwg:UserIdentity>
```

```
13 <urwg:JobName urwg:description="">UR test
14 </urwg:JobName>
15 <urwg:Status urwg:description="">Done</urwg:Status>
16 <urwg:Host urwg:description="">fingrid.boeblingen.de.
    ibm.com</urwg:Host>
17 <urwg:CpuDuration urwg:description="">PT0S
18 </urwg:CpuDuration>
19 <urwg:WallDuration urwg:description="">PT0S
20 </urwg:WallDuration>
21 <urwg:StartTime urwg:description="">
    2007-11-20T10:59:28Z</urwg:StartTime>
22 <urwg:EndTime urwg:description="">
    2007-11-20T10:59:28Z</urwg:EndTime>
23 <urwg:SubmitHost urwg:description="">fingrid.
    boeblingen.de.ibm.com</urwg:SubmitHost>
24 <urwg:Queue urwg:description="">Condor</urwg:Queue>
25 <urwg:Disk urwg:description="" urwg:storageUnit="KB">
    10000</urwg:Disk>
26 </urwg:UsageRecord>
27 </urwg:UsageRecords>
```

3.2 Metrics

Metrics are the measurable values gathered from the resources and represented in the UR document. For the purpose of our implementation, we used the standard metrics of the UR recommendation as well as a set of custom metrics as extensions. The custom metrics offer capabilities that were necessary to our use cases but that are not provided by the recommendation. They relate to the categorization of clients and resources and to the representation of the cost for the resource usages.

4 FinGrid Rule-based Billing

The purpose of this component is to provide an interface for generating usage bills. Billing is the process of generating bills from the resource usage data using generally a set of predefined billing policies. The bill can be in real money or it can use more abstract notions depending on the Grid site general policies. We should note here that the billing service does not preconize the use of a specific economic model. In fact it is independent from the economic model to be used. FinGrid Billing permits the definition, storage and manipulation of billing rules through a set of services. The rules are expressed in either a declarative rule language or simpler (but less expressive) *business* languages. They can be updated and deployed at run-time without any need to recompile or restart any part of the application.

4.1 Representation

We use knowledge representation to describe the different business rules. By using a knowledge based representation, we are constrained to use a logic-based language. Knowledge-based systems can be viewed at a symbolic level, or a knowledge level[New82]. Representation language formalism lies at the knowledge level, where we are concerned with the expressive adequacy of the language and its entailment relation. Logic being the study of entailment and rules of inference, the tools of formal symbolic logic are ideally suited for a knowledge representation system[BL04]. The choice for a logic rule language is dictated by the following criteria: entailment characteristics, expressiveness, Objects representation, Computational complexity, adequacy, extensibility, standard compliance.

4.2 Inference Engine

The inference engine is a reasoning system that keeps its actual knowledge in a database like structure called the *working memory*. The working memory gets updated in real-time with the changes in the system state. The inference engine's task is a three steps cycle[BL04]:

- recognize the rules that are applicable, i.e. rules whom prerequisites are a bill computation for instance.
- resolve conflict among the resulting rules.
- act accordingly by changing the working memory and firing the appropriate actions.

It is called inference engine because it matches facts with the rules to *infere* actions. The facts are stored in the working memory whereas the rules are stored in what is called *production memory*. Facts maybe added or removed from the working memory at run-time depending on the data received. A system with a large number of rules and facts may find at a certain time several rules to be true for a specific working memory state. Chances are that some of those rules would conflict (shutdown the database server vs. keep a high availability for premium clients to access the database). The inference engine needs then to implement conflict resolution strategies.

There are three types of inference engines. Forward-chaining types, backward-chaining types, and hybrid inference engines. Forward-chaining is *data-driven*. Facts are added in the working memory, and the inference engine looks for applicable rules with a conditional part being true, then adding the result to the working memory and evaluating again the rules against the facts until no new rule is fired. Backward-chaining is *goal-driven*. The inference engine is given a goal to reach and looks for rules with results matching the goal. Our business rules are deductive rules and foreward-chaining engines are the most appropriate for this type of rules. We are using Drools[Dro] as our inference engine. Drools is production rule system using an enhanced implementation of the Rete[For82] algorithm. Rules can be written in Drools Rule Language (DRL) or, using *expanders*, Drools provides the possibility to use Domain Specific Languages (DSL) by defining the

language semantics. We defined two DSL (technical DSL and natural DSL) that we are using in conjunction with the more complex DRL.

4.3 Billing Rules

We support two types of billing schemes: duration-specific and one-off cost. The duration-specific type applies the billing rule based on the usage duration whereas the one-off cost type is concerned with the proper usage of the resources. The following is an example of a duration specific rule written in the technical domain-specific language, it specifies 0.0002 unit (Euro, Pounds...), for every second of usage of the described resource and user type:

```
when EVENT = "VM Assignment", CLIENT_TYPE = "Platinum",
     RESOURCE_TYPE = "BLADE Type 4",
     RESOURCE_AGE < 240 * 60 * 60,
     SERVICE_LEVEL = "Platinum" then
      COST_PER_SECOND = 0.0002
```

A similar one-off cost rule in the natural DSL:

```
when the event is "VM Assignment" using a resource type
      "BLADE Type 4" and
     the user type is "Platinum" then the one of cost
      is 15
```

The second rule written in DRL:

```
1 rule "rule_2"
2 when
3 e:BillingEvent(event="VM Assignment",
   resourceType="BLADE Type 4", clientType="Platinum");
4 then
5 OperationResult fact = new OperationResult();
6 fact.setCost(10);
7 fact.setMessage("rule_2 asserted successfully");
8 e.setOperationResult(fact);
9 update(e);
```

The technical DSL as well as the natural DSL have very limited semantics and are intended to rules managers that are not programmers. The DRL is much more powerful and offer more expressiveness.

5 Architecture

5.1 FinGrid Accounting Architecture

Figure 1 shows our general architecture. Using the FinGrid portal, the user can submit jobs through the Community Scheduler Framework (CSF) or directly to a specific resource using GRAM. The collectors take care of gathering usage data and storing it in the records repository through the exposed FinGrid Accounting Interface.

5.1.1 Accounting Portal

The accounting portal is a web application (Figure 2) built using AJAX. It serves as a front-end to the FinGrid accounting service. It offers various management functions and operations on the usage records as well as the support for XPath for advanced queries. The accounting portal is integrated with the billing interface. Users can mark records for billing from the accounting portal.

5.1.2 FinGrid Accounting Service

For our accounting interface, we implemented OGF's Record Usage Service (RUS) [ANM05]. RUS is a stable OGF draft defining a basic infrastructure for accounting and auditing, it specifies the service interface to normalize operations on the accounting information of the resource usage as described by the Usage Record specification. The UR document can be maintained either centrally or in a distributed fashion. Our accounting service interface permits the upload of usage records or the extraction of necessary information and possibly the aggregation of resource usage data. It normalizes the operations on Usage Records documents that are stored in a persistent XML database.

5.1.3 Collector

We use collectors to gather accounting information from the local resources. A Collector is resource environment-specific agent that collects accounting data generated at the level of the Grid resources to which it belongs. This data is then submitted to the Grid Accounting Service in the form of UR documents. There is no proper standard for usage logs at the level of the resource manager. The collectors' task is to extract the relevant data from the generated logs and transform it into compliant OGF UR-WG document. For every Grid node, a collector agent instance is created whenever there is a job assignment, and the usage data is automatically collected and dumped to the RUS server. Currently we support Unix fork and OpenPBS. We should note here that some resource manager such as Platform LSF[LSF] provide logs directly in the UR format.

5.1.4 UR Repository

The RUS standard does not specify how the UR should be physically stored and leaves this decision to the implementers. We opted for the most natural way to store XML doc-

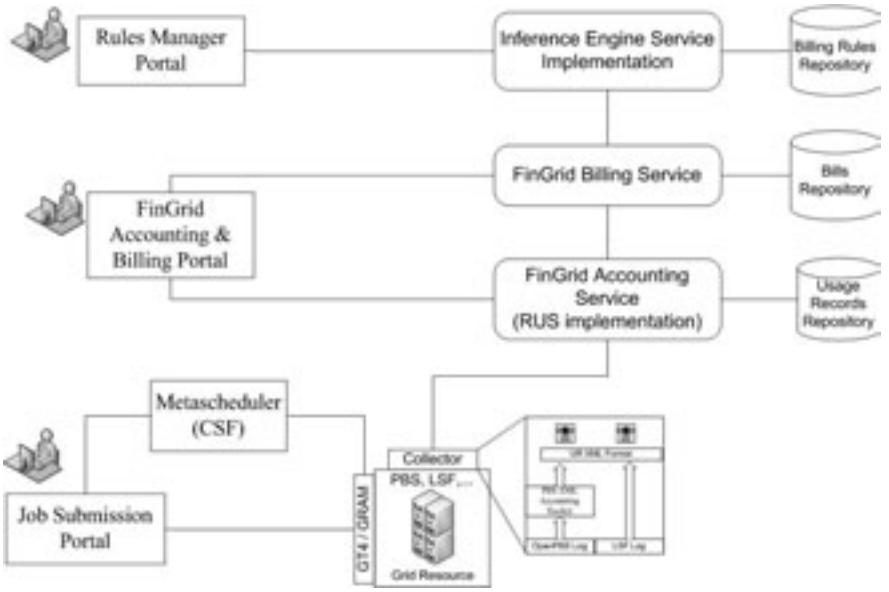

Figure 1: FinGrid general architecture

uments which is a native XML database. Our choice settled on Xindice[Xin] for the persistent storage. The advantage of using a native XML database is that we don't have to worry about mapping our XML document to a specific data structure to store in a normal RDBMS for instance. We store documents as XML and we retrieve them as XML. Xindice supports XPath 1.0 for querying XML documents which is very handful for aggregating results from different stored Usage Records, and XUpdate 1.0 for updating XML documents. Our implementation supports WS-Security as security layer for authentication and authorization.

5.2 FinGrid Billing Architecture

Our billing model separates the logic of the billing service from the data or the rules and uses a *hot-deployment* scheme where users can manage the billing rules and deploy them without restarting the billing service. This separation along with the possibility to write rules in more user-friendly languages, make our services accessible to a wide audience of users. The kind of users that would generally specify the billing policies but would not be necessarily a programmer or a computer savvy.

5.2.1 Billing Portal

In the billing portal we can mark single or multiple usage records for billing, generate usage bills and export the bills as XML files. However, sometimes we would need a higher

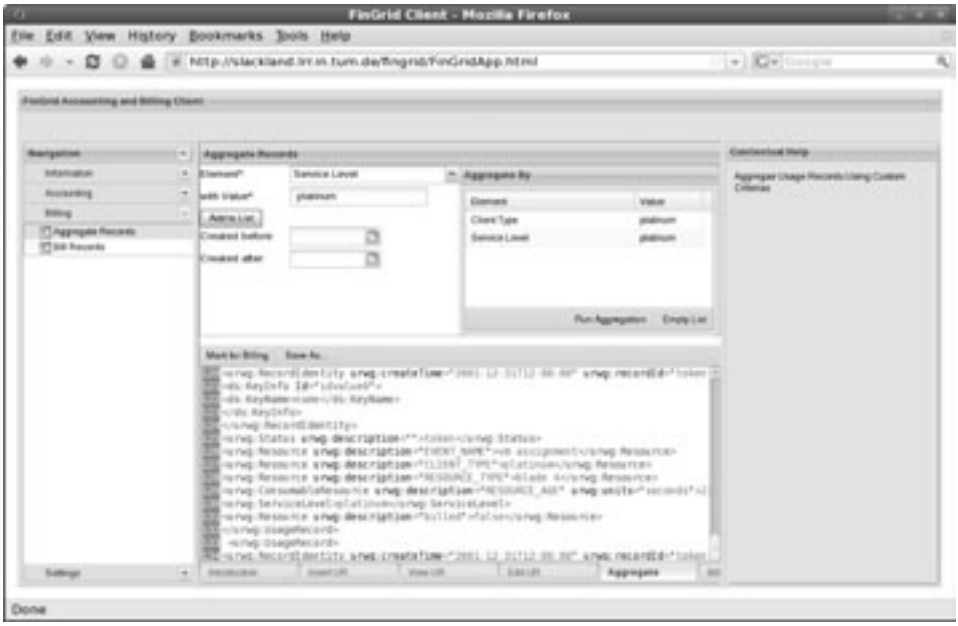

Figure 2: Aggregation support in the accounting portal

degree of control over which data to be billed. Our implementation is powerful enough to support aggregation of usage records over any combination of usage metrics elements as well as ranges of times. It also supports aggregation of usage data using complex XPath queries, giving the user a high degree of liberty in composing usage records. Records of virtual organizations or some specific parts of job can be then created using aggregation for a detailed billing document.

5.2.2 FinGrid Billing Service

The billing service provides port types to generate bills from usage data according to predefined billing rules. The records can then be marked as billed. We can also bill records created from aggregation results and save the aggregation for later reference. Our billing service implementation is a front end to the Drools inference engine. It parses the usages records and extract the relevant usage metrics for the billing process. It feeds this data to the inference engine working memory and fires the evaluation process. The bills generation process is not automated and needs to be initiated. However, we do have a command line version of our billing service that can be used with cron for instance for automated and scheduled bills generation.

5.2.3 Inference Engine and the Rules Manager Portal

As mentioned earlier, we are using Drools as our inference engine and we extended Drools BRMS to offer a management portal for the FinGrid billing rules. Users can write or edit rules written in any of the supported languages and deploy them without the need to recompile or restart the application, thanks to the separation between the application logic and the billing rules.

6 Related Work

Various work has been done in the field of Grid accounting and billing. The Distributed Grid Accounting System (DGAS)[ABG+] intends to implement resource usage metering, accounting and resource pricing in a distributed Grid environment. It supports decentralized banking structure where the billing occurs before the job submission. It also supports billing using various billing metrics. However, they tend to use proprietary solution and protocols for representing and exchanging accounting data. The Grid Accounting Service Architecture (GASA)[BB03] developed within the context of the Australian GRIDBUS project is another related work that supports different payment strategies (post-payment, pre-payment and pay as you go). It also supports billing using different billing metrics. It has, however, a centralized billing server and does not also adhere to the use of standards. The last example is the Swedish SweGrid Accounting System (SGAS)[SGE+04], an OGSA-based accounting architecture with decentralized banking structure. It supports a service-oriented architecture with an implementation of the UR recommendation. However, it only supports one metric (clock time per node) for the billing and is unable to track usage data on heterogeneous resources.

7 Future Work

In FinGrid, everything is a service but the Grid node themselves. As future work, we intend to change that by representing the Grid resources as services for ease of composition and management. We already identified the important points to achieve this and the approach that we should follow. It is clear to us that we need to describe the resources as services and provide a way to easily manage those services.

7.1 Describing Resources as Services

Web services are stateless application components, which are not suitable to describe and interact with Grid resources being logical or physical (servers, storage media,...) that need to maintain a state. Web services need therefore to define custom means to preserve state, discover other resources and interact with them. Standardization efforts were made to tackle this issue. The most prominent is the Web Services Resource Framework (WSRF)

set of specifications by OASIS [Ban06]. The WSRF provides a general solution using web services to an originally specific problem: describing and representing Grid resources. Another relevant feature of WSRF is that it brings a solution for management of a resource lifetime, faults and properties. A resource that is described in this way is called a WS-Resource. Rendering resources as WS-Resource and decomposing software component into services is the first step toward an SOA enabled management architecture with all the advantages that it can bring such as ease of management, adaptability and automation.

7.2 Managing Services

Considerable work has been done to define an architecture to manage web services, the most notorious are WSDM [BVWS06] and WS-Management [MMR08]. WSDM stands for Web Services Distributed Management and is composed of two sets of specifications: WSDM-MUWS, Management Using Web Services (MUWS) and WSDM-MOWS, Management Of Web Services (MOWS) specifications. MUWS specification defines how can we expose any resource as a manageable resource and is built on top of WSRF and WS-Notification [GHM06]. Seeing the importance of those new specifications, the consortia behind them decided to reconcile WSDM and WS-Management specifications into a single standard for management of IT resources using Web services [ea06].

8 Conclusion

We have presented the architecture and implementation of the FinGrid accounting and billing services based on open standards and recommendations. Our architecture is serviceoriented and modular. We used a rule-based approach for the billing service. This approach permits to detach the service control from the actual code enabling the business users to change the service behavior without the intervention of an IT staff, thus, enhancing greatly the application adaptability in a world where business rules may change on a daily basis. Our system is currently deployed at our academic and industrial partners for evaluation and testing.

References

[ABG+] C. Anglano, S. Barale, L. Gaido, A. Guarise, G. Patania, R. Piro, F. Rosso, and A. Werbrouck. The Distributed Grid Accounting System (DGAS). *http://www.to.infn.it/grid/accounting*.

[ANM05] J. Ainsworth, S. Newhouse, and J. MacLaren. Resource Usage Service (RUS) based on WS-I Basic Profile 1.0. *UR*, August 2005.

[Ban06] T. Banks. Web Services Resource Framework(WSRF) - Primer v1.2. *Official Committee Specification*, May 2006.

[BB03] A. Barmouta and R. Buyya. GridBank: a Grid Accounting Services Architecture (GASA) for distributed systems sharing and integration. *Parallel and Distributed Processing Symposium*, April 2003.

[BHF03] F. Berman, A. J. G. Hey, and G. Fox. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd, 2003.

[BL04] R. J. Brachman and H. J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufman, 2004.

[BVWS06] V. Bullard, W. Vambenepe, K. Wilson, and I. Sedukhin. Web Services Distributed Management. *Official Committee Specification*, August 2006.

[Dro] JBoss Drools. http://www.jboss.org/drools/.

[ea06] J. Antony et al. WSDM/WS-Man Reconciliation. *IBM Report*, August 2006.

[For82] Charles Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.

[Fos02] I. Foster. What is the Grid? A Three Point Checklist. *GRIDToday*, July 2002.

[GHM06] S. Graham, D. Hull, and B. Murray. Web Services Base Notification 1.3 (WS-BaseNotification). *Official Committee Specification*, October 2006.

[LSF] Platform LSF. http://www.platform.com/.

[MLM+06] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, and Rebekah Metz. OASIS Reference Model for Service Oriented Architecture V 1.0. *Official Committee Specification*, August 2006.

[MLMJM07] R. Mach, R. Lepro-Metz, S. Jackson, and L. McGinnis. Usage record - Format Recommendation. *UR*, 2007.

[MMR08] R. McCollum, B. Murray, and B. Reistad. Web Services for Management (WS-Management) Specification. *DMTF*, 2008.

[New82] A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[SGE+04] T. Sandholm, P. Gardfjell, E. Elmroth, L. Johnsson, and O. Mulmo. An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers. *Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 15–19, November 2004.

[Xin] Apache Xindice. http://xml.apache.org/xindice/.