Dynamic Service Reconfiguration and Enactment Using an Open Matching Architecure

van Splunter, Sander; Brazier, Frances M. T.; Padget, Julian; Rana, Omer

Publication date
2009

Document Version
Author accepted manuscript

Published in
Proceedings of the International Conference on Agents and Artificial Intelligence, Porto, Portugal

DYNAMIC SERVICE RECONFIGURATION AND ENACTMENT USING AN OPEN MATCHING ARCHITECTURE

Sander van Splunter, Frances Brazier
Intelligent Interactive Distributed Systems, VU University Amsterdam, The Netherlands
svsplun@cs.vu.nl, frances@cs.vu.nl

Julian Padget
Department of Computer Science, University of Bath, UK
jap@cs.bath.ac.uk

Omer Rana
School of Computer Science, Cardiff University, UK
o.f.rana@cs.cardiff.ac.uk

Keywords: Automated adaptation, matching, enactment.

Abstract: An architecture for dynamic reconfiguration of complex services, in which the enactment is automated, and the matching of services is not limited to a pre-determined set of matchers and repositories, is presented. The proposed architecture consists of three, previously developed, components: the CoWS template-based reconfiguration service, the Knoogle MatchMaker service, and the Triana workflow enactment engine. This architecture has the following innovative aspects: 1) automated adaptation of complex services, which is more flexible than existing approaches based on replacing failing instances of services within a workflow, 2) use of heterogeneous components that may be both local and distributed, and 3) dynamic selection of matchers and repositories.

1 INTRODUCTION

Web services provide uniform access to software capability with well-defined interface descriptions, and provide some limited options for semantic annotation. Many times, however, a single web service does not provide the precise functionality required for a specific application, therefore web services often need to be combined to form a complex service. If one web service fails, the continuity of the complex service is threatened. Instance-based replacement approaches offer one possible solution, but are limited to the availability of *exact* (function and interface equivalent) replacement services.

Current research allows automated handling of failing services, for example, based on late binding, but is often limited to instance replacement. The number of services available is large, but dispersed over different repositories, as different communities build and collect services. Even the design and deployment approaches may vary, for example, between Service Oriented Architecture/Grid and Peer-to-Peer based deployments.

The approach presented in this paper focuses on automated template-based reconfiguration of annotated web services, using (i) a discovery infrastructure with dynamic matcher service selection, repository selection and service policy specification, and (ii) a workflow enactment engine capable of handling heterogeneous components. This combination provides a fully automated reconfiguration service that can be used without human intervention. The resulting architecture enables: (i) automated service reconfiguration based on constraints that have been previously defined by a user; (ii) an *open* matching architecture that allows multiple matching mechanisms to be used, on a set of user-defined repositories—this is particularly useful to enable application-specific matchers to coexist alongside generic matchers based on term syntax; (iii) generality through integration with a workflow enactment engine, so that the approach can be combined as a component within an existing workflow. The remainder of this paper is organised as follows: section 2 briefly describes relevant research for the three areas addressed: web service reconfiguration, matchmaking and workflow enactment. Section 3 describes the proposed architecture. Finally, section 4 discusses the potential of the architecture, and future work.

2 RELATED WORK

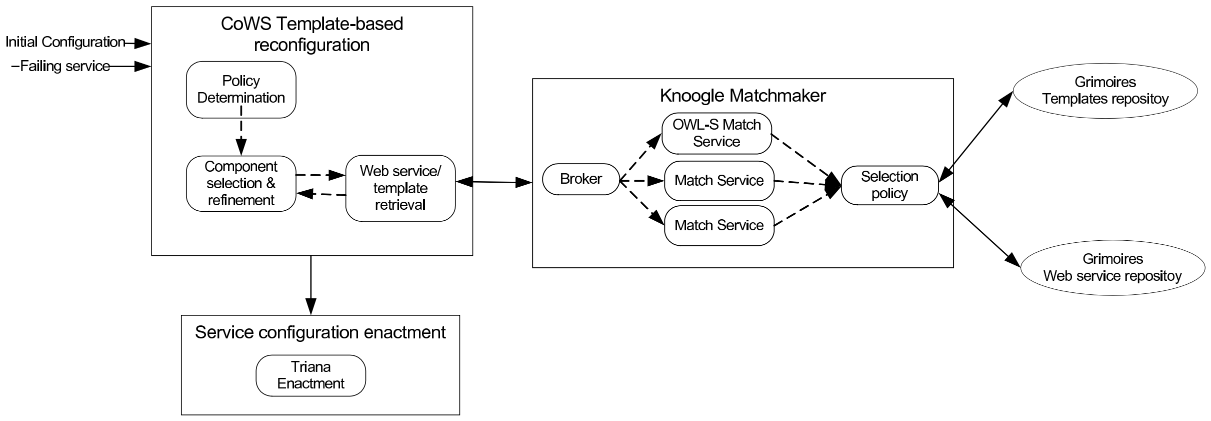
Three main areas of research are related to this paper: web service reconfiguration, matchmaking, and workflow enactment. A brief description of the current state of art is presented for each.

Web Service Reconfiguration The Web Service Reusable Flexible Flows (ReFFlow) project (Karastoyanova and Buchmann, 2004b) focuses on extending the meta-model of web service compositions to enable adaptability at run-time. A secondary goal for ReFFlow is automated creation of complex workflows using templates (Karastoyanova and Buchmann, 2004a). Supported adaptations are instance replacement, parameterised adaptation, or structural re-design. Adaptation in ReFFlow is limited in scope, because of the reasoning required to understand the effect of a service replacement with reference to the overall workflow. The ASTRO framework (Pistore et al., 2005) supports a planning approach to automated service composition based on abstract BPEL4WS processes, and composition requirements. The framework supports initial configuration, reconfiguration, monitoring, and automated instrumentation of the services. Similar to ReFFlow, the reasoning on the effect of a replacement of a service in relation to the properties of the complex workflow is limited. Cardoso and Sheth (Cardoso and Sheth, 2002) address the integration of new web services into existing workflows. Web services in this project are blackbox components with associated Quality of Service (QoS) parameters specified in DAML-S (the precursor to OWL-S). Component composition is based on workflow integration, and supported by abstract Service Templates, a self-defined local repository and discovery service. This approach requires human intervention for adaptation for the refinement of abstract Service Templates. Neither the matching architecture or the enactment is open. The MoSCoE framework (Pathak et al., 2006) is a goal-based, design time approach to composition, supporting both sequential and parallel composition. Services are modelled as Symbolic Transition Systems. MoSCoE starts with an abstract description of the desired service, provided by a human user, and supports the user in refining the description to a realisable BPEL workflow. The semi-automated nature limits its use for dynamic reconfiguration.

Matchmaking Matchmaking is used to discover dynamically suitable web services given service properties (which may also include QoS properties). In the literature generic brokerage mechanisms are described that use syntactic or semantic techniques for this purpose, or a combination of both. Some of the earliest systems, enabled by the development of KIF (Knowledge Interchange Format) (Genesereth and Fikes, 1992) and KQML (Knowledge Query and Manipulation Language) (Finin et al., 1994), are SHADE (Kuokka and Harada, 1996) operating over logic-based and structured-text languages and the complementary COINS (Kuokka and Harada, 1996) that operates over free-text using well-known term-first index-first information retrieval techniques. Subsequent developments such as InfoSleuth (Nodine et al., 1999) apply reasoning technology to the advertised syntax and semantics of a service description, while the RETSINA system (Sycara et al., 2003) has its own specialised language influenced by DAML-S and uses a belief-weighted associative network representation of the relationships between ontological concepts as a central element of the matching process. While technically sophisticated, a particular problem with the latter is how to make the initial assignment of weights without biasing the system inappropriately. A distinguishing feature of all of these systems is their monolithic architecture, in sharp contrast to GRAPPA (Veit, 2003) (Generic Request Architecture for Passive Provider Agents) which allows for the use of multiple matchmaking mechanisms. GRAPPA utilises a conventional multi-attribute clustering technology to reduce attribute vectors to a single value.

Workflow Enactment Workflow enactment involves managing the coordinated execution of web services. A variety of representation schemes encode the connectivity between services ranging from web service standard specifications such as WS-BPEL (Jordan and Evdemon, 2007) to specialist languages that are application specific, such as Condor DAGMan (Tannenbaum et al., 2002). Once such a connectivity graph has been described (often referred to as an "abstract" workflow), it is now necessary to manage its execution – either on a single machine or on distributed set of machines – often undertaken either by a single scheduler (often managing assignment of services to time slices on a single resource) or via some distributed group of schedulers. Hence, workflow scheduling is a kind of global task scheduling as it focuses on mapping and managing the execution of interdependent tasks on shared resources that are not directly under its control. The workflow scheduler/enactor needs to coordinate with diverse local management systems taking account of their local configuration and policies (Yu and Buyya, 2005).

Our approach uses an open matchmaking architecture, enabling application specific matching to be used alongside general purpose matching. Integration of this with the templates and workflow, render the approach more generalisable and adaptive.



Initial Configuration
Failing service
CoWS Template-based reconfiguration
Policy Determination
Component selection & refinement
Web service/ template retrieval
Knoogle Matchmaker
Broker
OWL-S Match Service
Match Service
Match Service
Selection policy
Grimoires Templates repositoy
Grimoires Web service repositoy
Service configuration enactment
Triana Enactment

Figure 1: Architecture for dynamic service composition and enactment.

3 ARCHITECTURE

The architecture (Figure 1) combines (i) the CoWS template-based web service reconfiguration, (ii) the Knoogle matchmaker, and (iii) the Triana enactment engine. These three components are linked together through a description of services expressed in OWL-S. The reconfiguration process is started after the detection of service failure in the complex service. It is activated with a description of the initial complex service, and a pointer to the service that failed. Based on the requirements posed to the failed service and to the initial complex service, the CoWS reconfiguration uses the Knoogle matchmaker to identify a replacement service, which is subsequently enacted by the Triana workflow engine. After a brief introduction of OWL-S in section 3.1, each of the three components in the architecture are described in sections 3.2, 3.3, and 3.4.

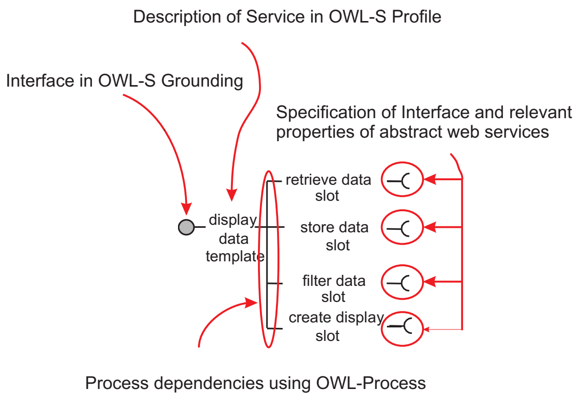
3.1 OWL-S

OWL-S (Martin et al., 2004) is an ontology for annotating web services, based on the Web mark-up language OWL (McGuinness and Van Harmelen, 2004). OWL-S describes a main service using three different sub-ontologies: the Service Profile, Service Model, and Service Grounding ontology. The Service Profile defines what the service does, in terms of service type, input, output, pre- and post-conditions. The Service Model defines the internal workings of a service. A Composite Process in a Service Model defines a process composition of sub-processes to which references are defined, together with the control flow specified with pre-defined control constructs. The Service Grounding links the abstract description of the service to actual implementation details, such as message exchange formats and network protocols so that automatic invocation of the service is possible.

3.2 CoWS Web Service Reconfiguration

CoWS is an automated template-based approach to reconfiguration of complex web services, in which the scope of the adaptation is adjustable. In addition CoWS enables reasoning on the effect of the replacement of a service in relation to the properties of the complex service (van Splunter et al., 2008).

A template describes a single-level composition of web services. This is defined as one or more slots, a control structure, and an associated template description. A slot is a placeholder for a web service or template, and defines a set of requirements for that component. The control structure defines the conditions for activation of the slots. The template description describes the functionality, behaviour and structure of the template, and dependencies between the slots.



Description of Service in OWL-S Profile
Interface in OWL-S Grounding
Specification of Interface and relevant properties of abstract web services
retrieve data slot
display data template
store data slot
filter data slot
create display slot
Process dependencies using OWL-Process

Figure 2: Template description in OWL-S.

A simple extension (Richards et al., 2004; Sabou et al., 2003) has been defined to specify templates in OWL-S, as illustrated in Figure 2. Every template has an OWL-S Profile, describing what this combination of slots would do, if used and refined. The OWL-S Service Model is defined as a CompositeProcess with slots defining abstract sub-processes. Each slot is defined as a subclass of Process, and requirements for each slot are included in this specification. A template description in itself has no Grounding Model. The Grounding is added when a complete configuration is created, based on the services and templates inserted in the template.

In our architecture, the CoWS service is activated given an initial complex service and an identifier of a failed service. CoWS then determines which slot was filled with this service. Based on the slot definition, functional and non-functional requirements are determined for a replacement service that needs to be created. A configuration process is started, which makes use of Knoogle, whilst providing a selection policy, the matcher and the repositories from which web services are to be retrieved. The CoWS process ends when an adapted template-based web service configuration is created which: (i) satisfies all requirements in the slot of the failing service; (ii) has no open slot in the resulting configuration, and (iii) violates none of the slot requirements in the resulting configuration. As shown in figure 1, CoWS supports policy determination, component selection and refinement, and web service/template retrieval.

The **policy determination** process chooses a policy specifying the strategy used for candidate selection. Examples of these policies are:

- *Favour service above templates.* If both services and templates fulfil the requirements, choose a service (the objective being to reduce the expected complexity of the configuration);
- *Fewest number of slots is preferred.* If a choice between templates is needed, choose the template with the fewest number of slots (the objective being to reduce the complexity of the configuration);
- *Most refinement candidates.* If a choice between templates is needed, choose the template with the highest number of refinement candidates (the objective being to increase the likelihood of success).

These selection policies are linked to the policies given to Knoogle. The **component selection and refinement process** is responsible for the overall configuration process, in which a replacement service is created for the failing service. The replacement service is either a single web service, or a complex service composed using templates. Initially, a slot is focused on for refinement, and the requirements associated with the slot are retrieved. These requirements and the selection policies are used to invoke web service/template retrieval. If a match is found, then the configuration process refocuses, until no open slots are left. If no match is found, the process backtracks, as a result of which it may mark a larger part of the initial configuration as failing. If no alternative services and/or templates are available at the highest node of the initial configuration, the configuration process fails. **Web service/template retrieval** prepares the requests to Knoogle to discover web service and template descriptions that match OWL-S based Profile descriptions of the services/templates that will fulfil slot requirements. The request includes the selected policy.

3.3 Knoogle matchmaker

Knoogle offers a brokerage framework that can be used to deploy partially or fully-configured brokers that query multiple service repositories, employ multiple matching services and apply pre-defined or bespoke selection policies. The matchmaking and brokerage process is characterised in terms of three essential actions: 1. where to find descriptions of entities to match against; 2. how to match the query against a description; and 3. how to choose between the matched descriptions. To achieve this, Knoogle receives four parameters from the CoWS service (see Figure 1):

1. A query specifying the requirement in XML (by default). However, the actual structure will be application-specific, since it must be compatible with whatever the matching service(s) expect. In this case it will be an OWL-S expression.
2. A list of URLs of UDDI-compliant repositories. These are the repositories to search when processing a query. This application currently works with two instances of Grimoires (Moreau, 2008) repositories, one containing semantic descriptions of services and one containing templates.
3. A list of URLs of match services. These are the services used to compare a query against a candidate service or template description from a repository. A match service must return a number in the range [0,1] to indicate the degree of match. This data is stored in the Knoogle broker as an RDF triple relating the match service, the degree of match and the candidate service. Currently, we utilise one similarity service (matcher), namely OWLSM (Jaeger et al., 2005).
4. A selection policy, specified as a query over the

```
input  : a query $q$
input  : a set of repository URLs R = $\{r_1 \dots r_i\}$
input  : a set of matcher URLs M = $\{m_1 \dots m_j\}$
input  : a selection policy $sp$
output: a service URL or set of service URLs
var    : TS, a set of $\langle m_k, [0,1], s_l \rangle$ triples
1 foreach r in R do
2     foreach s in r.services() do
3         foreach m in M do
4             TS = TS.insert(triple($m, m(q,s), s$))
5         end
6     end
7 end
8 return TS.query($sp$)
```
Algorithm 1: The core Knoogle algorithm.

RDF triples generated by the match services. The query language is triple-store dependent. The current version of Knoogle uses the Sesame(Openrdf, www) triple store and so queries may be specified in any of the three languages supported by Sesame (RDQL, RQL and SeRQL). The policies used here determine either the most appropriate service/template, or a ranked list of services and/or templates, depending on the CoWS query.

The matching process of Knoogle is as follows: the broker iterates over all the services/templates in all the repositories, computing a match for each one against the query using each one of the matchers, and then applies the selection policy to determine the "best" candidate(s). This process is formalised in Algorithm 1. We emphasise that the algorithm captures the *requirements* for the matching process, in that it shows each matcher being invoked (serially) on *each* query/service or query/template pairing. The current proof-of-concept implementation does precisely this, but clearly for large numbers of heavily-populated registries better technology, taking advantage of caching and parallelism, is needed to enable scalability.

3.4 Triana Enactment

After the CoWS reconfiguration creates an adapted complex service, the resulting service configuration needs to be enacted, for example as implemented by the combination of CoWS and the SMDS workflow manager in (van Veelen et al., 2008). However, SMDS is limited to strictly controlled environments. On the other hand, Triana[1] (Majithia et al., 2004) is a problem-solving environment designed for both the creation and execution of workflow graphs. Execution of these graphs can be done both locally and via a distributed enactment process (using Triana servers). Enabling enactment of reconfigured services in Triana allows integration of heterogeneous components and both local and distributed management of the execution of the services. To enable the execution of complex template-based services in Triana, a wrapper creates workflow graphs from the template-based web service configurations submitted to Triana. The wrapper utilises the control structures of templates, which are expressed in the ServiceModel of the OWL-S description of the templates. Currently a simple wrapper is used, supporting sequential activation.

[1] `http://www.trianacode.org/`

4 DISCUSSION

Combining a reconfiguration service with a discovery system and a workflow enactment engine is described. The architecture is designed to be flexible: supporting workflow adaptation, extensible from a matcher perspective, and generalisable through its use in a workflow. One of the challenges for future research is to increase robustness and scalability of the reconfiguration and matching process (for instance, the ability to pre-select repositories to search rather than search all user defined repositories). One option is to extend the configuration and matching environment to support parallel processing, another is to extend the system to support a more extended set of policies including heuristic-based policies. Interaction between policies within the three different systems has also yet to be explored.

Integrating three systems in the manner described makes the approach more complex to use, as it is generally required for a user to specify: (i) service description; (ii) selection policy; (iii) choice of suitable matchers; (iv) list of repositories to search. However, some of these can be pre-configured—such as (ii), (iii) and (iv). Currently, approaches to automated service composition (as outlined in section 2) are limited, either due to their lack of adaptability to specific application domains, or their overall complexity of use. By allowing different levels of system configuration, and enabling some of these to be pre-specified (not by an end user, but an application/system administrator), we believe it is possible to balance complexity of use with generality of the overall approach.

ACKNOWLEDGEMENTS: The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project (`http://www.icis.decis.nl`), supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. The authors are also grateful for the support offered for this research by NLnet (`http://www.nlnet.nl`), and through the Open Middleware Infrastructure Institute (`http://www.omii.ac.uk`) managed program (project Knoogle). The authors also wish to thank Pieter van Langen of VU University Amsterdam for his extensive input and Neil Chapman of Cardiff for his contribution to implementation of workflow enactment systems.

REFERENCES

Cardoso, J. and Sheth, A. (2002). Semantic e-workflow composition. Technical report, LSDIS Lab, Computer Science Department, University of Georgia.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). Kqml as an agent communication language. In *Proceedings of 3rd International Conference on Information and Knowledge Management, pp. 456-463*.

Genesereth, M. and Fikes, R. (1992). Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University. Available from `http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps`.

Jaeger, M. C., Rojec-Goldmann, G., Mühl, G., Liebetruth, C., and Geihs, K. (2005). Ranked matching for service descriptions using owl-s. In *Kommunikation in verteilten Systemen (KiVS 2005)*, Informatik Aktuell, Kaiserslautern, Germany. Springer.

Jordan, D. and Evdemon, J. (2007). Web services business process execution web services business process execution language version 2.0. WS-BPEL. Available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Karastoyanova, D. and Buchmann, A. (2004a). Automating the development of web service compositions using templates. In *Proceedings of Geschäftsprozessorientierte Architekturen Workshop at 'Informatik 2004'*, Ulm, Germany.

Karastoyanova, D. and Buchmann, A. (2004b). ReFFlow: A model and generic approach to flexibility of web service compositions. In *Proceedings of International Conference on Information Integration and Web-based Applications and Service*, pages 27–29, Jakarta, Indonesia.

Kuokka, D. and Harada, L. (1996). Matchmaking for information integration. *Journal of Intelligent Information Systems*, 6(2–3):261–279.

Majithia, S., Shields, M. S., Taylor, I. J., and Wang, I. (2004). Triana: A graphical web service composition and execution toolkit. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–524. IEEE Computer Society.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. (2004). Bringing semantics to web services: The OWL-S approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, Califonia, USA.

McGuinness, D. and Van Harmelen, F. (2004). Owl web ontology language overview, world wide web consortium (w3c) candidate recommendation. february 10, 2004.

Moreau, L. (2008). Grid RegIstry with Metadata Oriented Interface: Robustness, Efficiency, Security". Available at http://twiki.grimoires.org/bin/view/Grimoires/.

Nodine, M., Bohrer, W., and Ngu, A. (1999). Semantic brokering over dynamic heterogenous data sources in infosleuth. In *Proceedings of the 15th International Conference on Data Engineering, pp. 358-365*.

Openrdf (www). Sesame: Rdf schema querying and storage. Available via `http://www.openrdf.org`. Last accessed: February 2007.

Pathak, J., Basu, S., Lutz, R., and Honavar, V. (2006). Parallel web service composition in : A choreography-based approach parallel web service composition in MoSCoE: A choreography-based approach. In *4th European Conference on Web Services (ECOWS '06)*, pages 3–12, Zurich, Switzerland.

Pistore, M., Marconi, A., Bertoli, P., and Traverso, P. (2005). Automated composition of web services by planning at the knowledge level. In Kaelbling, L. P. and Saffiotti, A., editors, *IJCAI*, pages 1252–1259. Professional Book Center.

Richards, D., van Splunter, S., Brazier, F., and Sabou, M. (2004). *Composing Web Services using an Agent Factory*, volume 13 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 229–252. Springer Verlag.

Sabou, M., Richards, D., and Splunter, S. v. (2003). An experience report on using daml-s. In *Proceedings of WWW 2003 Workshop on E-Services and the Semantic Web (ESSW'03), Budapest, Hungary*.

Sycara, K., Paolucci, M., Van Velsen, M., and Giampapa., J. (2003). The retsina mas infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1–2):29–48.

Tannenbaum, T., Wright, D., Miller, K., and Livny, M. (2002). Condor: a distributed job scheduler. pages 307–350.

van Splunter, S., van Langen, P., and Brazier, F. (2008). Decentralized performance-aware reconfiguration of complex service configurations. In *Proceedings of the Performance for Peer-to-Peer Systems (P4P2P) workshop*. University of Warwick.

van Veelen, J., van Splunter, S., Wijngaards, N., and Brazier, F. (2008). Reconfiguration management of crisis management services. In *The 15th conference of the International Emergency Management Society (TIEMS 2008)*.

Veit, D. (2003). *Matchmaking in Electronic Markets*, volume 2882 of *LNCS*. Springer. Hot Topics.

Yu, J. and Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3):171–200.