Grant Agreement N° 215483

Title:	*Use Case Description and State-of-the-Art*
Authors:	*Philipp Leitner (TUW), Florian Rosenberg (TUW), Ivona Brandic (TUW), Schahram Dustdar (TUW), Franco Maria Nardini (CNR), Gabriele Tolomei (CNR), Fabrizio Silvestri (CNR), Kyriakos Kritikos (UoC), Jean-Louis Pazat (INRIA), Francoise André (INRIA), Olivier Barais (INRIA), Pierluigi Plebani (PoliMi), Attila Kertész (SZTAKI), Gábor Kecskeméti (SZTAKI), Zsolt Németh (SZTAKI)*
Editor:	*Zsolt Németh (SZTAKI)*
Reviewers:	*Dimka Karastoyanova (USTUTT)*
	Fabrizio Silvestri (CNR)
Identifier:	*PO-JRA-2.3.1*
Type:	*Deliverable*
Version:	*2*
Date:	*28 November 2008*
Status:	*Final*
Class:	*External*

Management Summary

The aim of this document is to introduce the most important aspects and the state-of-the-art of self-* infrastructures and service registry, binding and invocation.

The property commonly referred as self-* is a collection of one or more reflexive properties expressing the ability of changing some aspects of the working behaviour of a computing entity. They are surveyed as self-optimisation and self-healing in a numerical simulation example; autonomic brokering; dynamic self-deployment of services; dynamic adaptation, self-optimisation, self-healing and self-configuration in a mobile environment.

Software systems built on top of Service-Oriented Architectures (SOA) use a triangle of the three operations "publish", and "bind" in order to decouple the participants in the system. Key elements of the survey are Service Discovery (centralised vs. distributed, .keyword-based, signature-based, semantics-based and context-based approaches), QoS-based service discovery, Dynamic Binding and Dynamic Invocation.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany

Published S-Cube documents

These documents are all available from the project website located at http://www.s-cube-network.eu/

PO-JRA-1.1.1: State of the art report on software engineering design knowledge and Survey of HCI and contextual Knowledge

PO-JRA-1.2.1: State-of-the-Art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of SBAs

PO-JRA-1.3.1: Survey of quality related aspects relevant for SBAs

PO-JRA-2.1.1: State-of-the-art survey on Business Process Modelling and Management

PO-JRA-2.2.1: Overview of the state of the art in composition and coordination of services

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-CUBE materials are available from URL: http://www.s-cube-network.eu/

Contents

Chapter 1

Introduction

1.1 The purpose of the document

This document is aimed at surveying two important topics of service infrastructures: self-* service properties and service discovery. The two fields are discussed in two separate chapters and are not necessarily covered in full details rather, those topics are selected or highlighted that are most relevant to the work of this workpackage. In accordance with the Description of Work, the aim of this document is to understand the two main fields by selected use cases and surveying the current state-of-the-art related to the use scenarios. It is important to remark that the use cases presented in the document are illustrative examples of the state-of-the-art survey. In particular, they are not intended to be part of any future research roadmaps nor serve as general use cases for the entire S-Cube project. The presented use cases reflect the work that has been done by the different partners, they also summarize the experience they have gained from their previous work on topics related to the service infrastructures. In this way, the document is intended to provide a basis for common understanding of the current state and trends and will serve as a starting point for projecting the vision of the workpackage.

1.2 Self-* Service Infrastructure

The property commonly referred as self-* is a collection of one or more reflexive properties expressing the ability of changing some aspects of the working behaviour of a computing entity. Self-* can be translated to self-configuration, (and/or) self-optimisation, (and/or) self-healing, (and/or) self-protection in most cases but there are various further self-* properties like self-aware, self-adjusting, self-adapting, self-organized, self-recovering, just to mention a few. The aim of autonomic computing is to incorporate most of the self-* properties into computing systems however, current solutions rarely exhibit more than one or two self-* functionalities.

The driving force behind developing self-* functionalities is identified as the complexity of integrating large-scale heterogeneous computing systems into a single one [1]. The most notable trends nowadays in this direction are Grid and mobile computing. Accordingly, the chapter on Self-* Service Infrastructures is organised around these two computing platforms. The design space of self-* services is extensive and cannot be covered in a taxonomy-like manner in this report. Instead, some *representative cases* are chosen to introduce and demonstrate various aspects of self-* issues, their use cases and the best practices. Grid computing and self-* issues in grid environments are introduced by self-optimisation and self-healing numerical simulation examples in section 2.1. Autonomic brokering plays a crucial role in establishing self-* behavior in Grid computing hence, it is introduced in the next section. Similarly, dynamic self-deployment of services that is a unique technique necessary for establishing some aspects of self-* behaviour and it is described in details in section 2.3. Finally, dynamic adaptation, self-optimisation, self-healing and self-configuration are introduced in a mobile environment via

multimedia and transactional examples.

1.3 Service Registry, Discovery, Binding and Invocation

The other major topic in this document is the description of the state-of-the-art in the three basic functionalities that are part of service lifecycle. Namely, *service registration*, *service discovery*, and *service binding and invocation*. These functionalities are directly related to the three fundamental operations in Service-Oriented Architectures: ”publish”, ”find”, and ”bind”. Essentially, service registry and discovery techniques are necessary for decoupling clients from service providers. They, basically, implement publish and discover operations. The reason why decoupling is important is because in such a way clients can freely roam between business partners. Additionally, services and service based applications may alter in time that must be supported by changes in service contracts and protocols, accordingly. Hence, version management is also necessary to support multiple versions of services, allowing clients to dynamically choose between.

In chapter 3 registry and discovery techniques for SOA and web services are introduced. Best practices for registries are categorized as centralized or decentralized according to an *architectural* approach. Similarly, state-of-the-art of discovery mechanisms are introduced in groups as keyword-based, signature-based, semantics-based, context-based and quality-based approaches. This section is concluded by a detailed comparison of the most significant works contributed to this field so far. Next, issues of dynamic binding and invocation are surveyed in section 3.2. Finally, chapter 3 is concluded with an overview of version management in service infrastructures.

1.4 Methodology

The selection of use cases and compilation of the examples constituting the state-of-the-art was largely based on personal knowledge and experiences in past and related projects. This procedure consisted of standard searching in digital libraries (Google Scholar, ACM, IEEEXplorer). Papers have been selected to reference based on (1) citations according to Google Scholar, (2) relevance to the topics discussed, (3) novelty as perceived by the person reading the papers and (4) importance of the medium.

1.5 Connection to other WPs

This workpackage of S-Cube generally provides the infrastructure some of the other WPs are based on. Hence, some of the research topics in our WP listed below is the foundation of ’higher-level’ research workpackages. They are briefly mentioned here with more relevant details in the appropriate sections:

- JRA1.3 and JRA2.2 explicitly need possibilities to *monitor events*, which is conceptually provided by the infrastructure.
- JRA1.2 provides *methodologies for monitoring*, that could be applied at the infrastructure level.
- The notion of *adaptation* also appears in JRA1.2. Adaptation is a subset of possible self-* properties.
- *Service deployment* may serve as a basis for composition level self-adaptation studied in JRA 1.2.
- The *quality model* and the negotiation process that are necessary for *autonomic brokering* are thoroughly studied in JRA1.3.
- JRA1.3 has already investigated the efforts in identifying, and defining the QoS attributes that is necessary for *quality based service discovery.*

- *Service discovery* is essential for *service composition* dealt with in JRA2.2.
- For *adaptive compositions* investigated in JRA2.2, dynamic invocation of services and mediation of incompatible interfaces and business protocols is essential.
- *Service compositor initiated deployment* may offer the possibility of modifying the service infrastructure according to the needs of the composition as investigated in JRA2.2.

Chapter 2

Self-* Service Infrastructure

Self-* is a pun. Self-* does not exist on its own rather, it is a collection of various properties that are present in autonomic computing systems and intended to realise self-managing or self-governing functionalities. Although the notion has been around earlier, most works trace back the roots of autonomic computing to the initiative by IBM in 2001 [2] published later as the 'Vision of Autonomic Computing' [1]. The integration of heterogeneous distributed systems into a single computing environment that may be extended beyond any boundaries introduces new levels of complexity approaching the limits of human capabilities. This is identified as a major obstacle for further progress and a source of a potential software crisis [1]. As a solution (and as they claim, probably the only solution) IBM proposed the notion of autonomic computing - systems that are able to manage themselves based on the high-level objectives given by human administrators. The term *autonomic* comes from a biology metaphor: the autonomic nerve system governs many of the low-level yet vital functions of the human body without any conscious intervention. Similarly, autonomic systems are assumed to maintain and adjust their operations in face of changing components, workloads, demands, external conditions and failures, both accidental or malicious [1].

Types of self-* behavior. The Vision of Autonomic Computing [1] identified four self-* properties commonly referred as self-management: self-configuration, self-optimisation, self-healing, self-protection. Self-configuration is related to installation, configuration and integration of large complex systems in an autonomic way. Self-optimisation is an effort made by the autonomic system to find ways of improving its operation, be more efficient in performance and cost.

Self-healing is an ability to detect, diagnose and repair localized problems caused by bugs or failures in software and hardware components. Self-protection is a defense against large-scale, correlated problems arising from malicious attacks or cascading failures. It also means anticipating and avoiding problems based on early symptoms.Since then follow-up works distilled further self-* categories as self-awareness, self-monitoring, self-adjusting, self-anticipating, self-adapting, self-critical, self-defining, self-destructing, self-diagnosis, self-governing, self-organized, self-recovery, self-reflecting, self-simulation and self-stabilizing [3] and potentially others. (Needless to say, some of these categories have overlaps or correlation; some are even orthogonal to others and do not establish clear disjunct categories.) Hence, self-* or *selfware* refers to some of the above mentioned functionalities that are necessary to realize (some aspects of) autonomic behavior. Definitely, a 'self-* system' or a 'self-* service' does not mean realizing *all* of these properties – it would be close to human intelligence.

Levels of self-* behavior. Autonomic properties may be present at various levels from individual computing components, e.g. a single computer or simply a disk drive to large-scale federated computing systems. These levels require different techniques to ensure self-* behavior. Higher levels involve more flexibility and dynamism. Ultimately, autonomic behavior would be expressed as declarative high-level goal-oriented statements that are decomposed into lower level detailed requirements at elements of the

autonomic system on the fly. Service-oriented concepts play a fundamental role for autonomic computing nevertheless, some augmentations are necessary.

Lifecycle of the self-* services. An autonomic element (e.g. a self-* service) has its own lifecycle that goes through design, test and verification; installation and configuration; monitoring and problem determination, recovery; upgrading; uninstallation and replacement. Each such stage comprises some special features. *Design, test and verification* of autonomic elements require managing the relationships between elements, needs and preferences along with capabilities; specification of high-level goals and mapping them to lower-level actions. *Installation and configuration* of autonomic elements involve service publication and discovery in directory services. *Monitoring and problem determination* are essential functionalities for autonomic systems as they may be continuously performing adaptation, self-optimization, reconfiguration and self-healing. An important aspect of monitoring is ensuring the quality of required and provided services. *Upgrading* of the services can be installing newer versions or creating entirely new service instances in an autonomic way. Obviously, autonomic services may be engaged in many simultaneous activities representing different stages of the lifecycle. Hence, *management of the lifecycle* is necessary in order to prioritize and schedule the activities so that elements can maintain a consistent view.

Similarly, the relationships between autonomic elements also have a lifecycle. *Specification* must include output services that a certain autonomic element can produce and input services it may require. An autonomic element must be able to *locate* input services whereas it must be locatable by other services. *Negotiation* encompasses several procedures that can be involved in obtaining a service once a provider was found. After the negotiation phase, *provision* allows access to the elements' resources. *Operation* means the utilisation of the negotiated and provided services, resources and finally, *termination* ends the agreement between parties that can reallocate their resources for other uses.

As it can be seen, the design space of self-* services is quite large. In this short and rough discussion above a three dimensional space was spanned (although, obviously, not every item in this imaginary vector space makes sense.) To cover the complete taxonomy of self-* services is out of scope of this deliverable – if not impossible. This document contains some representative use-cases and their states-of-the-art. Nevertheless, the series of S-Cube deliverables contain some strongly related material. The life-cycle of services is introduced in more details in deliverable JRA1.1.1 [4]. Monitoring is an essential functionality for establishing self-* services. They are covered in depth in deliverable JRA1.2.1 [5] and only mentioned here briefly.

2.1 Self-* Properties in Grid Computing

There are various attempts to define Grid computing. The concept was first overviewed and analysed in-depth in "The Anatomy of Grid" [6] and later refined in "The Physiology of Grid" [7]. They define Grid computing as a "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [6]. Another often cited and probably more practical approach for describing Grid computing known as the "Three point checklist" [8] that qualifies Grid as (i) coordinates resources that are not subject to centralized control; (ii) using standard, open, general-purpose protocols and interfaces; (iii) to deliver nontrivial qualities of service.

Expressing the common understanding of a researchers' community, CoreGRID defined Grid computing as: "A fully distributed, dynamically reconfigurable, scalable and autonomous infrastructure to provide location independent, pervasive, reliable, secure and efficient access to a coordinated set of services encapsulating and virtualizing resources (computing power, storage, instruments, data, etc.) in order to generate knowledge." `http://www.coregrid.net/mambo/content/view/2/25/`

To give an informal characterisation according to these definitions, computational Grids are based on large-scale resource sharing that typically geographically distributed, and span multiple administrative domains. In Grids, the virtual pool of shared resources is dynamic and diverse, since the resources can

be added and withdrawn at any time according to their owners discretion, and their performance or load can change frequently over time. The typical number of resources is of the order of several thousand or more. For all these reasons, a user has very little or no a priori knowledge about the actual type, state and features of the resources constituting the pool [9]. This characterisation clearly resembles the motivations of autonomic computing [1] as introduced in Section 2.

Resource allocation in Grids is often limited to batch scheduling. With batch scheduling, it is impossible to ensure that a given number of resources will be available at a given time. If we consider services that should run at any time it is merely impossible to ensure that they will get all the needed resources. Even if there exists load prediction algorithms that may improve the situation, there is no way to predict crash of machines. The study of mapping and scheduling algorithms however, is out of scope of this document.

On the other hand, when considering high performance simulations or services on the Grid, the runtime can be very long, so the probability to encounter a failure is close to one. The performance of computing nodes and the network may change abruptly due to sharing. Some components, services may undergo upgrades, their availability may change. An application must be able to adapt to the available resources. It can be concluded, that *some* autonomic behavior, e.g. self-healing, self-configuration, dynamic adaptation, self-deployment, is becoming an essential property in Grid services. Some examples are introduced in the followings.

2.1.1 Use Cases

Numerical Simulation Applications on Grids

Large scale numerical simulation applications enable the speed-up of the design of complex manufactured products, such as cars or aircrafts. These applications need high performance and high availability of resources.

Self-optimisation: Performance. Due to the dynamic nature of large-scale distributed systems in general, and the Grid in particular, it is very hard to design an application that fits well in any configuration. Moreover, constraints such as the number of available processors, their respective load, the available memory and network bandwidth are not static. For these reasons, it is highly desirable that an application could take into account this dynamic context in order to get as much performance as possible from the computing environment.

Self-healing: fault tolerance. Numerical simulations can run for very long time. Because they use a large number of machines there is a high probability that one more machine crashes or need some maintenance during the execution of a single program.

So, there is a need for “fault tolerance” that may be seen as a particular case of dynamic adaptation: it is the adaptation to the ”crash” of some of the allocated resources. However, things are not that simple. Fault tolerance encompasses two different problems: the recovery of the fault and the adaptation to the new situation. Only the second one really belongs to the problem of dynamic adaptation. Many strategies can be used to recover from faults. The closest one from dynamic adaptation is probably checkpointing in the sense it requires some “global” point where the execution state can be stored and then restart the application from that point if a crash occurs later on.

2.1.2 State-of-the-art of self-adaptation in Grid environments

In this section we have highlighted some projects as examples of projects providing self-adaptation or mechanismes suitable for self-adaptation in Grid environments. We do not intent to make a review of the numerous Grid projects here.

Self-adaptation can be made in several parts of the system: at the operating system level, at application level, at library level or in a middleware layer. We do not study here the operating system adaptation which is application-transparent. Some generic adaptation mechanisms can be included into the operating system so applications developers don't need to rewrite their applications to cope with the variations of the environment but the adaptation is not often very accurate.

Adaptation at the Application Level

Dynaco. The Dynaco model of self-adaptation [10] relies on a framework for Fractal components. Its goal is to separate the adaptation process itself from the user's code. The framework is composed of 3 parts: decider, planner and executor. These components provide the general functionalities for self-adaptation and the necessary interfaces. The coordinator is an optional part for parallel components. Content, actions, policy and guide are specific to the component. The decider input are events generated by external monitors. These monitors observe the environment or the state of the component itself. The decider relies on the policy to decide "what should be done". The policy can be a simple ECA rule based system (Event-Condition-Action) or a more sophisticated system such as a neural network. The planner provides a list of actions to the executor (such as: stop the service, move to another machine, restart). It relies on a guide that can be a simple collection of predefined plans or a more sophisticated planning system. The executor executes the actions provided by the plan on the user's code.

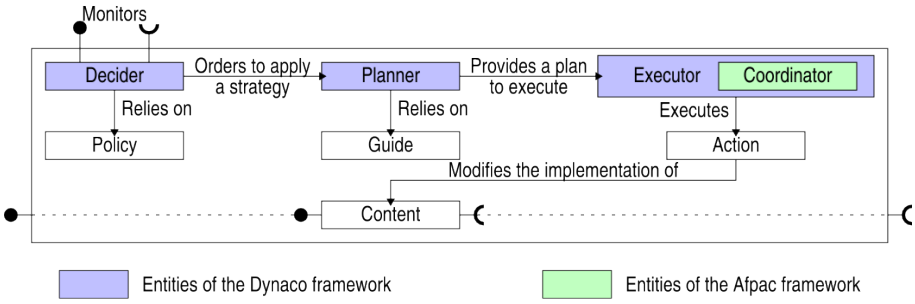

Figure 2.1: The Dynaco framework

ASSIST. ASSIST [11] is another framework that allows to describe applications with a coordination language as a graphs of modules interconnected by typed streams of data. There exist parallel modules called parmod which are used to coordinate a set of concurrent activities. The QoS assigned to an ASSIST component may be modified dynamically. In order to take this into account, the ASSIST framework dynamically adapts parallelism and process mapping. The adaptation mechanism relies on automatic user code instrumentation, and on a hierarchy of Application Managers.

GridCCM. GridCCM [12] is an extension for existing CCM (CORBA Components Model) implementation. Parallel CORBA components are defined as a collection of identical CCM components. They are expected to communicate thanks to an external mechanism, like for example via MPI (Message Passing Interface). On one hand, component models enables to dynamically assemble component through runtime API. However, it appears to be a low level solution as the behavior of the application (or part of it) is embedded into a component implementation. On the other hand, some component models support ADL. However, it does not provide an adequate level of abstraction for scientific applications nor it enables to capture the evolution of the assembly. In [13], the authors study this issue with the master-worker paradigm which can be seen as of static and a dynamic paradigm. In a steady state, the application may be described with a static ADL. However, two features have to be adapted to the environment: the number

of workers and the actual policy used to transfer request from masters to workers. The utilization of the collection concept enable to offer a simple model to application developers while allowing an efficient execution. The programmer builds a collection as a standard component except that its implementation is made with a (or several) component type(s). Collections can be connected to other components (or collections) as standard components. A collection does not define in advance (e.g. at programming time) the number of instances it will contain during its execution. For instance in the master-workers paradigm, the number of workers will not be fixed in advance; depending on the number of requests and on the number of available processors to execute the workers, their number can be adjusted. Also the pattern that is used to handle incoming requests and to distributed them to the workers can be dynamically adapted. This could range from a simple round-robin distribution if the number of requests and the load of the workers are low, to a complex distribution pattern using intermediary requests managers as in Diet [14]. Hence, these properties can be decided at deployment time or even better dynamically at execution time,to adapt the application. In order to implement self-adaptable master-worker programs, the authors relies on the Dynaco framework described here [10].

Adaptation at the Middleware Level

The main rôle of a middleware is to add the missing functionalities of the underlying operating system that are needed at the application level. Grid middleware is a basis for distributed and parallel applications because most Grids are based on an interconnection of machines running independent operating systems. So Grid middleware replace a “single system image system” by adding functionalities at the Grid level such as deployment, load balancing, scheduling, file transfer, identification.

Globus. The Globus Toolkit [15] is a set of libraries and programs for the management of Grid infrastructures. Globus has been used in many scientific projects such as TeraGrid and EGEE. The current version (GT4) is organized as a set of loosely coupled Web services. It provides tools for deploying, monitoring and executing distributed programs (GRAM), an implementation of GridFTP for file and memory transfers and a tool for service discovery. Within GT4, a framework allows the user to build and integrate new services. A “metascheduler” (GridWay [16]) has been developed and provides advanced scheduling capabilities including support for self-adaptive applications and support for the definition of new scheduling policies.

NextGRID. The NextGRID project [17] is a FP6 project that aimed at providing a vision and an implementation of Grid-based applications and Grid middleware for supporting an ”unlimited range of applications and business processes throughout their lifecycle”’. Both the model and the implementation are tightly related to OGSA and try to enforce a standardization. To achieve this goal, the project provided a ”Cookbook” as a guide for system designers and developers who want to implement a Grid.

The project adressed many challenges such as management related challenges (availability, service discovery, integration of legacy code, usage control over resources, interoperability, non-exclusive access to resources), performance related challenges (data rates, scalability, interactive access) and licensing.

Within the NextGRID proposed architecture, in order to ensure end-to-end QoS and a separation between Business level, Service Composition level and Infrastructure level, there is an SLA translation between each level service description. Each service having ”obligations” and providing ”guarantees”. There could be negotiations of SLAs in the proposed architecture but we have not seen any adaptation mechanism to achieve or to maintain QoS. An ”adaptive API” was designed but its goal is to provide a common programming ”language” to build and execute workflows. This is not related to dynamic or self adaptation in our vocabulary.

Condor. Condor [18] is a middleware for distributed execution of programs on a set of varying resources. It has been used for building Desktop Grids (e.g. using idle time on computers on a network) and

on larger Grids in conjunction with Globus. Condor dynamically adapts itself to the available resources and using a checkpoint/restart mechanisms is allows the adaptation of programs with no modification.

Grid5000. Grid5000 is a project from the French ACI GRID [19]. It is composed of a large set of clusters distributed at the national level in France. Grid5000 is not an operational system for running applications but a testbed for applications, Grid middleware and tools. Grid5000 allows self-configuration of applications because the user can choose is own software stack. These properties are similar to virtualization but are done at the lowest level, allowing high performances.

More details on Grid middleware can be found in deliverable [20].

Apples and GridWay. The dynamic characteristic of Grid architectures appears well accepted, as reflected by the presence of monitoring services such as Globus MDS. On the other side, job management usually considers that the set of resources allocated to one job remains constant for the whole lifetime of the job. Approaches such as AppLeS [21] and GridWay [16] show that even Grid applications can benefit from non-constant schedules. With those two projects, when the scheduler or the application detects that a better schedule can be found, the application checkpoints itself, then the job is canceled and resubmitted with new resource constraints.

GrADSolve. [22] is a RPC based environment to deploy and execute high performance parallel applications on Grids . The parallel application can be selected from an XML database by the problem specification, the input data of the client is uploaded to the machines and the corresponding application is executed. GrADSolve relies on Globus [15] and uses dynamic scheduling techniques to choose the resources used for the execution.

Dynamic Adaptation at the Library Level

SaNS. The SANS effort (Self-adapting Numerical Software) [23] aims to ease the design and use of efficient parallel numerical codes. In scientific applications, performance is a major concern but portability can not be ignored if one wants to benefit from the improvement of hardware. In this approach, numerical libraries are encapsulated into software components that can be selected automatically to give the best performances. Statistical techniques are used to predict performances and are used for the dynamic selection of a component. Selection of a component includes parametrization of the algorithms. This approach has also been used for fault tolerance purpose.

ProActive. The ProActive middleware [24] is a Java library, which aims at achieving seamless programming for concurrent, parallel, distributed, and mobile computing. ProActive relies on the concept of active objects remotely accessible as depicted in Figure 2.2. Results of remote methods calls and synchronization are handled by a mechanism known as wait-by-necessity [25].

Thanks to this model, self-* properties can be easily added to ProActive programs. ProActive middleware targets Grid programming at three different levels: Grid Infrastructure, Grid Technical Services, and Grid Higher Level Programming.

Grid.It. The Grid.It project [26] faces the problem of dynamically adapting parallel components. The followed approach is based on structured parallelism provided by high-level languages. The compiler generates code for virtual processes that are mapped to processing elements at run-time. An adaptation consists in dynamically modifying this mapping. This adaptation can be done within language constructs transparently to the developer. In this case, the consistency enforcement mechanism can rely on assumptions on the overall structure of the program as it is generated by a compiler.

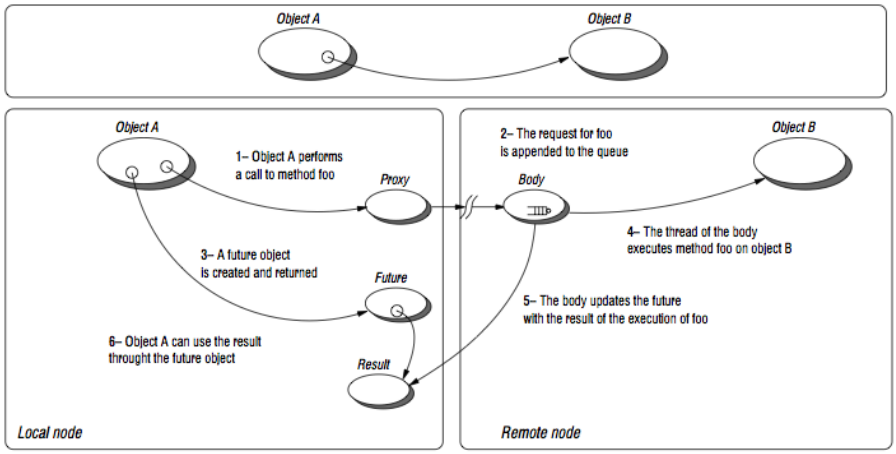

Figure 2.2: ProActive model

Kaapi. Kaapi is another middleware for multi-threaded parallel programming. It supports a macro dataflow language (Athapascan [27]). Thread scheduling relies on work-stealing: an idle processors "steals" a waiting task for execution in a queue. Because Athapascan programs are mainly recursive, stealing fine task generates many new tasks that the stealer can execute (or that another processor can steal). This framework is a basis for self-adaptation in the sense that the scheduling and allocation are not fixed. In [28], the authors have built a self-healing environment by connecting Kaapi and Dynaco.

PCL. PCL [29] focuses on the reflective programming tools for the dynamic adaptation of parallel applications. This project considers dynamic adaptation as modifications on a static task graph modeling the code and provides at runtime primitive operators on that graph. It gives a full framework for reflective programming that is useful to implement reactions. The problem of the coordination of the reaction and the execution threads of the functional code has been studied with PCL in [30].

Self-adaptive Middlewares

In contrast with the techniques described in Section2.1.2 middlewares in this section do not only provide functionalities to help the design of self-adaptable applications but are self-adaptable themselves. This means that such a middleware can adapt itself to the environment and/or to the application needs. Reflexive middlewares have served recently as foundation to build adaptable service-oriented application or service-oriented platform. Component based reflexive middleware is often based on three key concepts: components, component frameworks, and reflection. Both the application and the middleware platform are built from interconnected sets of components. Components and component frameworks provide extendable structure and functionality, and reflection offers the essential support for dynamic configuration and extensibility for runtime evolution and adaptation.

Several underlying component model exists as for example Fractal or OpenCom. We propose in this document to give a short overview of these component based reflexive middleware and to present safran a framework to build self-adaptive applications.

OpenCOM. OpenCOM is a general-purpose and language independent component-based systems building technology. Components are language-independent units of deployment that support interfaces and receptacles. Interfaces express units of service provision. Receptacles express units of service requirement and are used to make explicit the dependency of one interface on another (and hence one

component on another). Interfaces are expressed in terms of sets of operation signatures and their data types. Bindings are associations between a single interface and a single receptacle.

The basic concepts of OpenCOM are capsules, components and bindings. Capsules are containing entities that offer a component run-time (CRT) API for the loading, binding etc. of components. Components are language-independent units of deployment that support interfaces and receptacles (receptacles are required interfaces that indicate a unit of service requirement). Bindings are associations between a single interface and a single receptacle.

As mentioned, reflection is used to support introspection and adaptation of the underlying component/ CF structures. A pillar of OpenCom to reflection is to provide an extensible suite of orthogonal meta-models each of which is optional and can be dynamically loaded when required, and unloaded when no longer required. The meta-models manage both evolution and consistency of the base-level system. The motivation of this approach is to provide a separation of concerns at the meta-level and hence reduce complexity.

Safran. [31] is an extension of the Fractal component model (http://fractal.ow2.org) to support the development of self-adaptive components, i.e. autonomous software components which adapt themselves to the evolutions of their execution context. Safran is composed of three sub-systems on top of Fractal (see Figure 2.3):

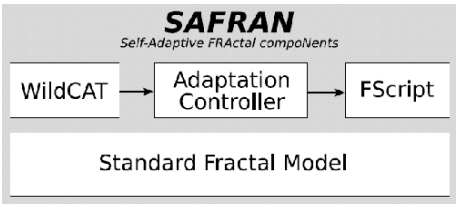

Figure 2.3: Safran architecture

FScript is a domain-specific language used to program the component reconfigurations which will adapt the application. It provides a custom notation which makes it easy to navigate in the components Fractal architecture and offers certain guarantees on the changes applied to the target application, for example the atomicity of the reconfigurations. WildCAT [32] is a generic toolkit to build context-aware applications. It is used by Safran policies to detect the changes in the application's execution context which should trigger adaptations. Finally, an adaptation Fractal controller binds FScript and WildCAT through the reactive rules of adaptation policies. These rules follow the Event-Condition-Action pattern, where the events are detected by WildCAT and the actions are FScript reconfigurations. The adaptation controller allows the dynamic attachment of Safran policies to individual Fractal components and is responsible for their execution. Safran can be reused in the context of ESB like PEtALS (`http://petals.objectweb.org/`) in order to provide reconfigurable and adaptable service oriented applications.

Reflective middleware for Mobile computing. In [33] the joint use of reflection and metadata is proposed in order to develop middleware targeted to mobile settings. Through metadata separation of concerns is obtained, that allows to distinguish what the middleware does from how the middleware does it. Reflection is provided to applications in order to inspect and adapt middleware metadata, that is, influence the way middleware behaves, according to the current context of execution.

Comparison

Table 2.1 summarizes the way self-properties are handled in the studied Grid environments. Projects are classified by ”level” according to the previous descriptions. ”Yes” in the Monitoring column states that monitoring have been studied or that monitoring tools are included in the prototype, Self properties lists the main kind of self-property handled. Decision rules is ”Yes” if there is a built-in support for including decisions rules to build an ECA (Event-Condition-Action) system. If parallel/distributed programs are handled, the corresponding column is ”Yes”. When it is not clear if the environment takes into account one of these programming paradigm (e.g. it is not explicit but it might be possible), we have prefered to put an ”?”. Implementation is ”Yes” when there is a known an available implementation, ”Spec” when only a specification is provided, ”proto” when a prototype have ben provided.

None of the tools/projects studied provide all the necessary functionalities to support adaptable and self-adaptable distributed services. As a conclusion, we can state that it is a challenging task to provide all these functionalities at any level.

Level	Tool/project	Monitoring	Self-properties	Decision rules	Parallelism	Distributed computing	Implementation
Application level	Dynaco	No	supports *	Yes	Yes	Yes	Yes
	Assist	Application level	supports *	Yes	Yes	Yes	Yes
	GridCCM	No	Optimizing	No	Yes	Yes	proto
Middleware level	Globus	Yes	supports *	No	Yes	Yes	Yes
	NEXTGrid	Yes	Support * (multilevel SLA)	No	?	Yes	Spec/Proto
	Condor	Yes	Scheduling and Healing	No	Yes	Yes	Yes
	Grid 5000	Low level	No	No	Yes	Yes	Yes
	Apples/GridWay	?	MetaScheduling	No	Yes	Yes	Yes
	GRaDSolve	Yes	Deployment and scheduling	No	Yes	Yes	Yes
Library level	SanS	?	Healing and performance	No	Yes	?	Spec
	ProActive	Yes	supports *	No	Yes	Yes	Yes
	Grid-It	No	supports *	Yes	Yes	Yes	Yes
	Kaapi	No	Healing and performance	No	Yes	Yes	Yes
	PCL	No	supports *	Yes	Yes	Yes	Yes
Self-adapting middleware	OpenCom	No	supports *	Yes	?	?	Yes
	Safran	Yes	Self-reconfiguration	Yes	No	?	Yes

Table 2.1: Support for self-* properties in Grids

2.2 Autonomic Broker Agents in Grids

Investigating the state-of-the-art self-* approaches in current Grid resource management, we can see that mostly agent-based techniques are used to bring autonomous properties into grid brokering. Grid Brokers are resource managers responsible for finding a suitable execution environment for a user application. In Grids, the executable programs are called jobs, and the execution machines are called computing resources. The role of the brokers is to match the user requests to the available resources taking into account the requirements of the jobs (e.g. memory, CPU speed requirement) and the properties and load of the resources (this is also called matchmaking or scheduling). The Grid and Agent communities are both pursuing the development of open distributed systems, though from different perspectives [34]. The Grid community has historically focused on interoperable infrastructure and tools for secure and reliable resource sharing within dynamic and geographically distributed virtual organizations, and applications of the same to various resource federation scenarios. In contrast, those working on agents have focused on the development of concepts, methodologies, and algorithms for autonomous problem solvers that can act flexibly in uncertain and dynamic environments in order to achieve their aims and objectives. In order to be effective, Grids must be imbued with flexible, decentralized decision making capabilities, and agents need a robust distributed computing platform that allows them to discover, acquire, federate and manage the capabilities necessary to execute their decisions. Agent-based solutions have already appeared in Grids, we will give a summary of state-of-the-art approaches in the next subsections. In the latest Grid solutions, service interactions are structured via Web service mechanisms, and thus all entities are services. While every agent can be considered a service (in that it interacts with other agents and its environment via message exchanges), not every Grid service is necessarily an agent (it may not participate in message exchanges that exhibit flexible autonomous actions). As Web and Grid development are converging, we can notice that Web and Grid techniques generate an environment, where the boundary of job and service requests is vanishing. This means that new Grid workflows will appear, in which a user incorporates general grid jobs, new grid service calls and even external web service calls. Brokers need to be able to distinguish these requests and need to communicate with additional services that perform service discovery and monitoring. In order to cope with such workflows, brokers need to be autonomous, more adaptable and fault-tolerant. In this report we are not intended to analyze intelligent or mobile agent solutions. We only focus on autonomic techniques already appeared in Grids. In the next subsection we introduce use cases in this area, then we summarize the relevant research works done so far in this field.

2.2.1 Use cases

To reveal the need for autonomous operations in state-of-the-art Grid Computing we refer two main areas that are in the focus of intensive research: establishing Grid Interoperability and bringing Service Level Agreements (SLA) into Grids. Brokers may support or rely on other services in order to perform a special task. For example, in section 2.3 we present a use case, where service deployment is carried out together with a service broker.

Grid interoperability is highly studied by various projects, OGF [35] has also published several standards to help this process. The goal is to interconnect the currently available production Grids in order to create a bigger computing power. A recent study in [36] discusses Grid interoperation, and introduces use cases for problems of different data management protocols, description languages and service interfaces. At the level of resource management several research groups have started to extend their resource brokering solutions to support various grid middleware services and serve different grid user groups - OGF has also started to investigate inter-broker communication. The global view of our use case for interoperable application execution is the following: a user submits its application to a broker, it finds suitable resources of different Grids for the jobs of the application, then it manages the execution of the jobs and returns the results. In order to point out aspects where self-* solutions are used or needed, we introduce a possible implementation of this use case. Since brokers are not able to access different Grid

resources, a higher level service is used as an entry point to Grids: a Meta-Broker [37], which performs a higher level brokering by matching user request to brokers of different grids. This matchmaking takes place under a high level of uncertainty (e.g. unknown number of competing jobs waiting for a resource), therefore clear descriptions and enhanced autonomous and self-aware operations are needed. When job requirements are matched to broker capabilities continuous monitoring is necessary (in terms of broker performance and their resources) in order to perform an adaptable and failsafe operation. The following subsection gathers solutions that solves some of these self-* aspects (adaptability, autonomy, fault tolerance and scalability).

Introducing SLA usage into Grid brokering is an inevitable step towards business Grids. Extending resource managers with agreement handling also affects the matchmaking process. A more detailed survey of current approaches can be found in [38]. Imagine a scenario, where a user signs an agreement with a broker (or with a meta-broker). He/she may require co-allocated resource usage, a certain deadline for job completion and offers a certain amount of money in turn. First a negotiation process is started, for which the broker should already have information about the available resources and their properties, furthermore some predicted availability. After the contract is made the resources should be reserved and the planned execution should be enforced even in a faulty environment. This operation needs high adaptability to react to the highly dynamic nature of resource availability, enhanced monitoring to notice resource failures and load unbalance, and self-awareness to cope with the fluctuations in the number of users. The quality model and the negotiation process to be followed in such a use case is thoroughly studied in JRA-1.3.

2.2.2 State-of-the-art

Though there is not much related work for autonomic brokering components in Grid systems, there are definitely several promising solutions that can be regarded as forerunners of autonomic Grid services.

Grid Agents. A combination of agent-based approaches for global grid load balancing is used in [39]. The broker agents cooperate with each other to balance workload using a peer-to-peer service advertisement and discovery mechanism. This work gives a global view on the combination of Grid and agent techniques, therefore we present a more detailed summary. It introduces a grid agent structure shown in Figure 2.4 is defined by three layers:

- Communication layer: Agents in the system must be able to communicate with each other or with users using common data models and communication protocols. The communication layer provides an agent with an interface to heterogeneous networks and operating systems.
- Coordination layer: The request an agent receives from the communication layer should be explained and submitted to the coordination layer, which decides how the agent should act on the request according to its own knowledge. For example, if an agent receives a service discovery request, it must decide whether it has related service information.
- Local management layer: This layer performs functions of an agent for local grid load balancing. This layer is also responsible for submitting local service information to the coordination layer for agent decision making.

Agents are organized hierarchically in a higher level global grid environment Figure 2.4. The broker is an agent that heads the whole hierarchy. A coordinator is an agent that heads a sub-hierarchy. Leaf-nodes are simply termed agents in this model. Brokers, coordinators and agents are differentiated by their position in the hierarchy but not by their functionality. Each node has the same capabilities and priority for service and could perform services on behalf of any other node. This homogeneous hierarchy provides a high-level abstraction of a grid environment. The agent hierarchy can represent

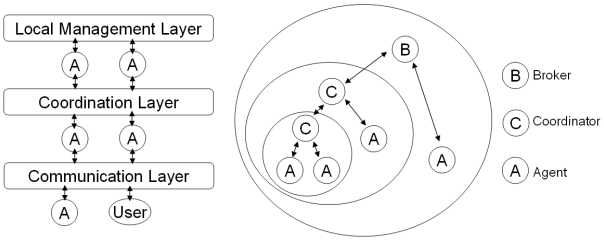

Figure 2.4: Grid agent structure and hierarchy

an open and dynamic system. New agents can join the hierarchy or existing agents can leave the hierarchy as appropriate. The hierarchy exists only logically and each agent can contact others as long as it has their identities. Service is another important concept: an agent provides a simple and uniform abstraction of the functions (clients, services and go-betweens) in the grid management system. The service information provided at each local grid resource can be advertised throughout the hierarchy and agents can cooperate with each other to discover available resources. In this way, a grid is a collection of multiple local grid resources represented by agents that are distributed geographically in a wide area. The brokering problem here is merged with the discovery of available grid resources that provide the optimum execution performance for globally submitted tasks. The service discovery process indirectly results in a load-balancing effect across multiple grid resources. An agent can also receive many service advertisements from nearby agents and store this information in its coordination layer as its own knowledge. In the coordination layer each agent records service information of the local grid resource. The local management layer is responsible for collecting this information and reporting it to the coordination layer. Each agent can record the service information received from its lower or upper agents in the hierarchy. There are basically two ways to maintain this information flow: data-pull and data-push, each of which has two approaches: periodic and event driven. Discovering available services is also a cooperative activity. Within each agent, its own service provided by the local grid resource is evaluated first. If the requirements can be met locally, the discovery ends successfully. Otherwise global service information is evaluated and the request dispatched to the agent, which is able to provide the best requirement/resource match. If no service can meet the requirements, the request is submitted to the upper agent. When the head of the hierarchy is reached and the available service is still not found, the discovery terminates unsuccessfully. The discovery process does not aim to find the best service for each request, but endeavors to find an available service provided by a neighboring agent. While this may decrease the load-balancing effect, the trade-off is reasonable as grid users prefer to find a satisfactory resource as fast and as local as possible. Both advertisement and discovery requests are processed between neighboring agents and the system has no central structure, which otherwise might act as a potential bottleneck.

RL Agent. The work in [40] applies multi-agent learning techniques to the problem of resource allocation in the Grid. Introducing learning allows the multi-agent system to adapt to changes, such as changes in resource capacities, resource failure, or introduction of new agents into the system. The multi-agent learning approach studied here relies on minimal monitoring capabilities to compare resources, only requiring that the agent obtains status signals for job requests issued by the same agent. They use a so-called reinforcement learning, which is a powerful framework in which an agent (e.g. a Grid user), learns optimal actions through a trial and error exploration of the environment and by receiving rewards for its actions. The reward function defines what the good and wrong actions are in different situations. The agent's goal is to maximize the total reward it receives. For a single agent in a stationary environment, the problem reduces to finding the optimal policy. In their multi-agent system, a large number of users submit jobs to one of the resources that are scheduled by a local scheduler according to local policies. The users (and brokers) are modeled as rational agents that try to maximize their utilities (complete

their jobs in the shortest possible time). The agents have no prior knowledge about the computational capabilities of resources: they utilize a simple reinforcement learning scheme to estimate the efficiency of different resources based on their past experience. After each submitted job, the agent updates the score of the corresponding resource. Resource providers are job-executing agents that offer resources to serve computational needs of user agents. Each resource provider agent is characterized by the type and amount of the resources under her disposal, as well as by the local scheduling/prioritizing policies. They use a simple learning mechanism for resource selection: for each possible action the agent keeps a score that indicates the efficiency of that resource in the past. For each new job, the agent chooses a resource which has the highest score, with ties broken randomly. After each completed job, the agent gets a reinforcement signal (containing the start-time and the endtime for that job), calculates the utility, and translates it into a reward for the chosen resource. Finally, the agent updates the scores.

AWG. The Autonomous Semantic Grid system architecture [41] aims at providing an improved infrastructure by bringing autonomy, semantic interoperability and decentralization in the Grid Computing for emerging applications. One realization of this system is the AgentWeb Gateway [41], which is a solution that facilitates two-way service discovery, service description transformation and communication protocol conversion among FIPA compliant Software Agents and W3C compliant Web Service without modifying these existing standards by acting as middleware. It uses protocol converters to make Software Agents, Web Services and Grid Services understand and call each-other. The solution converts between SOAP and ACL (Agent Communication Language), and between WSDL and DFAgentDescription using UDDI and DF (Directory Facilitator). This solution demonstrates how interoperation among different platforms and technologies can be established.

GSP. Though agent-based solutions are dominating, there are some other approaches to bring autonomic behaviors into Grids. The Grid Service Platform [42] is a middleware for autonomous resource sharing based on the priority of business operations. They approach the brokering problem taking into account that load changes cause fluctuations in the number of servers required, therefore Grid systems must flexibly allocate server resources. They use a Physical Resource Broker (PRB), which virtualizes physical servers, and brokers logical servers. PRB receives heartbeats from the node monitors set up in each physical server and maintains a list of alive servers. It also conducts a search of the database, which exists in the Grid information subsystem, to obtain a list of physical servers that meet the server requirements of a particular business operation. The Lifecycle Manager (LM) is used for requesting and returning additional logical servers according to the workload. An Arbitrator (ARB) can receive requests from all LMs and adjusting the allocation of logical servers. PRB virtualizes physical servers and brokers them as logical servers to ARB, while ARB arbitrates the requests from LMs and determines the allocation of logical servers. ARB diverts servers from low-priority business operation services and allocates them to high-priority business operation services that are requesting servers. On the other hand, a policy to dynamically change the priority can be set on LM to minimize the loss incurred by server deprivation. LM always notifies the latest priorities to ARB.

ASC. Another example of this approach is the BU-Grid [43] that uses a bidding mechanism for autonomic service compositions and selection in a Grid environment. The basic idea for bidding in this architecture is the following: if I denotes a particular service interface, E_i(I) denote the estimated service time of the i^{th} implementation for the service interface I, B_i(I) denote the value sent to the broker by the i^{th} implementation, where the value is the bid of using interface I. The broker that also acts as a search engine, first notifies each of the service providers that hosts the required service implementations. Being notified, each service implementation will make use of the current estimated service time E_i(I) (track record) as well as the current system load (for resource i: L_i) to compute a bid value and send the bid back to the broker, where $B_i(I) = (1 - L_i) * 1/(E_i(I))$. The broker then selects a service implementation according to the probability distribution. After the selected implementation finished the assigned job, it

will notify the broker of the result. The broker will then return the actual service time, and the estimated service time of the i^{th} service implementation will be updated.

GrADS. Taking a closer look on other self-* approaches in Grid Computing, we can barely find any services having some of these capabilities. Grid systems are forming: service-based approaches have only started to spread. Though adaptive operation is congenital in brokering services: since Grids have a highly dynamic nature, the load unbalances need to be handled by these resource managers, which requires adaptability. Going further, a more self-adaptive operation is presented in [44], which describes a framework for application migration in Grids (which is an important feature of fault tolerance) with the help of execution time prediction. In this solution the migration depends on the amount of increase or decrease in load of the resources, the point during the application execution lifetime when load is introduced into the system, and the performance benefits that can be obtained for the application due to migration. The framework has been implemented and tested in the GrADS system [45]. The ability to migrate applications in the GrADS system is implemented by adding a brokering component called Rescheduler to the GrADS architecture. The migrating numerical application (Migrator), the Contract Monitor that monitors the application's progress and the Rescheduler that decides when to migrate, together form the core of the migrating framework. This framework is primarily intended for rescheduling long running applications, and such scheduling policies are used that only suspend and migrate applications, when better systems are found. This solution can be a forerunner of self-healing operations.

Table 2.2.2 summarizes the self-* Grid brokering approaches described in this section.

Paper	**Adaptability**	**Autonomy**	**Scalability**	**Fault tolerance**
Grid Agents [39]	-	Yes	Yes	-
RL Agents [40]	Yes	-	-	-
AWG [41]	Yes	Yes	Yes	-
GSP [42]	Yes	Yes	Yes	Yes
ASC [43]	-	Yes	-	-
GrADS [45]	Yes	-	-	Yes

Table 2.2: Comparison of the self-* properties of state-of-the-art solutions in Grid Brokering.

Finally we would like to mention two trends pointed out by the recent Grid projects, namely OGF [35], NextGrid [17] and CoreGRID [46]: these are interoperability issues and the usage of Service Level Agreements (SLAs). Providing a certain level of quality of various services is becoming a key issue in several current Grid projects [47]. To guarantee some level of quality is far from easy in these heterogeneous, distributed and highly dynamic systems. Previously, most distributed computing provisions followed a so-called best effort operation: a service provider attempts to provide a consistent, available service but makes little or no guarantees on that provision. For example users are able to specify the minimum level of hardware required to run their job. These requirements are matched by the brokers to available resources or the request is queued until a resource with the specified requirements becomes available. However, within this type of service provisioning users may need additional guarantees apart from those capabilities of the resources the actual grid offers. For example, users may require urgent tasks to be completed by a certain point in time, or to be provided with a guarantee that a certain class of resources are pre- or co-allocated. At the moment, most brokering services agree and enforce these guarantees manually, but the automated usage of agreements is becoming more and more urgent. Therefore the goal of the brokering agents is shifting from job makespan minimization to the fulfillment of user requirements, which will require self-* capabilities.

2.3 Autonomic Service Deployment

Even though several software life-cycle models can be defined deployment is still the usual part of these models. Deployment is usually placed somewhere between implementation and maintenance. However in some cases software can be designed with deployment awareness, therefore deployment techniques can influence even the earliest stages of the life-cycle. In the self-* service infrastructure the automation of deployment can help the SBA to adapt the application both to the infrastructure constraints and the its current needs.

According to the Open Grid Forum's CDDLM Foundation ([48]) document the deployment can be defined as follows: "Deploying any complex, distributed service presents many challenges related to service configuration and management. These range from how to describe the precise, desired configuration of the service, to how we can automatically and repeatedly deploy, manage and then remove the service. Deployment description challenges include how to represent the full range of service and resource elements, how to support service "templates" (where some description files can be used later on as a base for future deployment), service composition, correctness checking, and so on. Deployment challenges include automation, correct sequencing of operations across distributed resources, service lifecycle management, clean service removal, security, and so on. Addressing these challenges is highly relevant to Grid computing at a number of levels, including configuring and deploying individual Web Services (including WS-RF and other dialects), and composite systems made up of many co-operating Web Services."

In this document it is assumed that the deployment is composed of the following basic steps. We call a component as the *deployment system* if it can accomplish at least one of these steps:

Selection is an operation choosing the appropriate (hardware and software) system to deploy the software product on. The result of the selection process is the target system where the deployment would take place. In the scope of self-* infrastructure selection is used to initiate the whole deployment process, therefore this step supports the automation of service placement.

Installation is dealing with the injection of the necessary software components to the target system. These components include the software product itself and its dependencies.

Configuration adjusts the installed software components to fit the target system's specific details.

Activation is the execution of the deployed software product. Performing the execution depends on the underlying system.

Deactivation might be required to enable maintenance related operations on the deployed software product like adaptation, update or decommission.

Adaptation makes a previously installed and configured software components fit to the current needs of the target system. Is crucial for self-* infrastructure since the adaptation step can be repeated several times to make sure the currently deployed service fits the requirements of the SBA.

Update replaces previously installed software components and then initiates a reconfiguration on them.

Decommission removes the unnecessary software components from the target system, then issues a reconfiguration task on the remaining software systems if needed.

Deactivation, adaptation, update and decommission are also maintenance related tasks, these deployment tasks help the service's maintainer and also the infrastructure's maintainer to modify the current deployments according to the current needs of the SBS.

2.3.1 Use cases

Automated deployment simply means that the deployment system offers most of its capabilities to the outside world through well defined interfaces. For example other services in the SBA can directly access the deployment system's adaptation interfaces when they would like to change the service configuration during its execution. Thus the automation of service deployment tasks comes from third party services whose quality requirements against the SBA cannot met without adjusting the application to their needs. Here we provide a list of third party services which can act as initiators for deployment. This list is built on well known components of the SBAs which can all make use of a deployment system in order to achieve their goals. These use cases are discussed from the initiator point of view, to see where extensions are necessary on the current solutions in order to provide use the automation interfaces. Since these initiators dealt with static environments (e.g. where all service instances are known prior executing a schedule) they cannot be applied here. In the next paragraphs we will discuss each of the use cases in more detail, focusing on the differences between the traditionally available systems and the use cases made available by involving a deployment system.

Service brokering: Upon request service broker (see section 2.2) defines execution schedules based on service availability and possible deployments

Service composition: The execution of the composition can be optimized by the compositor during composition time by considering further service deployments to reduce execution costs (for example deploying services in the composition closer to each other), or it is also possible to optimize in runtime when the compositor can deploy replacements for services which became unavailable or busy during execution.

Predictor engine: Predict future peak usages and deploy services to avoid them.

Self-replicating service: On faulty (or extreme) service operation services should initiate their renewal on other site(s)

Request for unknown service: A service container receives a request what it cannot process without deploying the service locally

A *service broker* can initiate a new deployment in case it receives a request to a heavily demanded (e.g. a service which is requested more frequently than a the average) service or to provide faster turnaround times. The brokering service which can initiate deployment is built on a service repository. Which helps to define a schedule to execute a service request taking into consideration both those sites where the service has been deployed and where it could be executed but has not yet been installed. The candidate sites can be selected based on the metadata stored in the repository next to the service code itself. If the deployed services are not available, it checks whether any of the candidate resource providers can deliver the service taking into account the deployment cost. This scenario can be fitted in the Open Grid Forum's ecosystem. And it is defined in the execution management system's initial reviews [49]. Brokers have a huge history and background behind them, however extending them to support this use case can reduce the infrastructure constraints they would face. While it is possible to offer entirely new scheduling algorithms which take the deployment capabilities of the infrastructure into consideration from the beginning, there is another approach which builds on top of the current scheduling techniques. This approach describes the deployment as any other task which can be scheduled. Therefore implementors of the new use case would require to add these deployment tasks to the schedule when they become necessary.

A *service compositor* or a BPEL engine should provide effective service placement (current service locations might not be useful for the composition's output). Since the compositor is a higher level service it can use composition information to make requirements against deployments. Here we list some examples where these decisions should be addressed in the composition level. Depending on service

estimated response times (which can be acquired from the GLUE schema [50] of the service provider) the compositor engine might decide to deploy a service while the composition is executed (because the engine actually knows the future needs of the service infrastructure). The compositor also has the possibility to deploy services in order to reduce network latencies between the different service calls. This is especially useful when the services in the composition are massively distributed but the individual service executions of the composition are short ones. Therefore, in this way service deployment offers the base for composition level self-adaptation (see JRA 1.2 for details). There are several available service compositor engines already (see JRA2.2 for details), however they are all lacking the feature where they can adjust the service infrastructure according to the needs of the composition. These compositors usually know the business workflow in advance, before all the workflow execution takes place. Therefore nodes that should be executed later on can be bound to such services that were deployed during the workflow execution. The new compositors can be extended with such workflow evaluation logic which can initiate service deployments for some of the business process's nodes. Or alternatively workflows can be extended prior their execution to incorporate a conditional service deployment node before critical service invocations. This conditional deployment node then can decide whether the infrastructure needs to be extended with a new deployment at the time when the workflow reaches the critical service invocation.

A *predictor engine* could detect service usage patterns, or identifying future peak service usages in order to prepare a possible deployment request to avoid unnecessary delays. Since service deployment usually takes significantly more time than invocation in overloaded periods the invocation times might grow rapidly. With a prediction engine the service deployments can be started before the need. There are two ways to predict the service demand, be the scheduler who knows about all the service invocations. Or use prediction logic based on past behavior and current need (from the information systems). This scenario supports both the load balancing approaches of scheduling and also the automated initiation of selection for the resource providers, and these are the two tasks which are making up the deployment preparation process. In case of automation the information system (like the MDS4 [51], or UDDI, see section 3.1 on page 30 for more details) is the most crucial part of the SBA. The information system has to provide up to date information about available services and pricing, and it should also support of registration of resource providers and their policies. These policies include restrictions about the services that can be hosted, and the price ranges within the resource provider would offer its resources for newly deployed services. The use case here is different from the available service provisioning systems for a simple reason: usually provisioning is built into already existing components like the scheduler or the compositor, however in this case the predictor engine initiates deployments and decommission independently from these components. As a result a dynamic service environment can be used in schedulers and compositors even though they are not even changed.

A *self-replicating service* ([52]) uses a self-monitoring infrastructure to determine when an erroneous situation lasts or higher load occurs locally. A service experimenting these issues can decide to lower its burden by deploying its exact copy somewhere else. The service usually takes this decision when self-reconfiguration was not successful since its health sensors still present problematic situations. E.g. the service reached the limitations of its hosting environment. Self-replicating service is the local version of the prediction engine in a sense that it will result a more dynamic service environment even though higher level SBA components are not changed.

Finally a service container might receive a *service request for a service not present* in the current context. Depending on site policies it can initiate deployment on itself. To support this scenario the container analyzes the service call, and tries to identify a service interface which provides the given call. In case multiple interfaces found they are filtered by the virtual organization offering them. If this is still not possible and semantic web services are used then semantic data might also be provided with the call, which could help identifying the required service to be deployed. Only those services are searched which belong to the same virtual organization the user is participating in.

2.3.2 Requirement analysis

This section presents the required components which should be present in a system capable to handle all the use cases mentioned before. These requirements are used to set up the classification presented in the next section.

A deployment system should support service repositories[53], where the service's code, configuration, activation and version details can be obtained. Repositories are necessary for the automation of deployment in order to offer sources of deployable services. If repositories would not exist then automated deployment systems should be able to collect the services code when deployment requests are made. Collection of the service code could mean acquiring the service and all of its dependencies from one of its running instances. This gets even worse when the same service has to be deployed several times, because the service's code has to be collected every time instead of just downloading it from a repository. Service code collection can be a long running task therefore its continuous repetition can reduce the overall performance of the service based system even though further deployments were initiated just to avoid performance drop.

Several repositories exist today. Since they have to be used in a service oriented system, they also should support service oriented interfaces for the create, read, update, delete (CRUD) and searching operations. Because of the different nature of the artifacts these repositories dealing with the service oriented interfaces might only need to support metadata operations such as versioning and presenting configuration options (see section 3.1 on page 30 for more details). For downloading large chunks of data the repository might offer transfer URLs, where the user of the repository can find the required data temporarily for that specific download operation. Another approach to fulfill this requirement is to provide a registry (holding all the metadata about the service) and link the items not suitable for storing in the registry. These links usually point to a file on a simple HTTP server. Even though this seems like a simple solution it raises security issues since the security has to be handled in both the registry and on the URL the registry links. With this link based solution it is possible to have different access rights on the registry entries and the linked remote entities, therefore to avoid these problems the registry and the linked entities have to be managed together manually. Otherwise it could happen that even though someone does not have access to a given registry entry, the linked item on the remote side could be accessible against the will of the registry's maintainer.

The automated deployment system should have a service, which can be used for deployments of services upon request (both manual and automated case). This service should cover the installation, configuration, and activation phase of the deployment process. Lower level interfaces already exist to support parts of this process. For example the Globus Virtual Workspace Service [54] supports installation, if the services are packaged for a virtualized environment. Packaging services for virtualization is proposed among others by [55]. In case lower isolation level is enough a container level installation solution can be used for hot service deployment [56]. Alternatively new invocation strategies can be introduced like in WSPeers [57]. These strategies can incorporate installation.

Different installation procedures require different activation solutions, because the installation usually determines the methods available for the activation of the service code. For example activating a service, which was delivered as a virtual machine image, would mean the startup of the virtual machine with that specific image. As another example the service container based deployments require the activation technique to use the service container's own activation code after the service is installed within the container.

The service supporting automated deployment should not just focus installation of the service, but also on its configuration. Configuration can be supported by implementations of the CDDLM specification [58]. CDDLM can easily support specific configuration systems, however it is not generic, therefore other, more generic interfaces should be used instead. Generic interfaces include WBEM ([59], which is not service oriented, therefore cannot be fitted entirely to a SBS). Oasis (Organization for the Advancement of Structured Information Standards) offers a full service oriented and generic configuration specification as WSDM, which can deal with the configuration of web services [60] and non-service

oriented components underlying SBSs [61].

A predictor engine should be also present in an automated deployment capable system which should operate on top of current SBSs alongside a service scheduler. The predictor engine detects usage patterns of services with the help of information systems and self-monitoring data. It is crucial that highly distributed systems do not spread service metadata e.g. availability, average response time, conforming WSDL frequently in order to lower the network overhead. Because of this reason information has to be gathered from globally and locally as Grid systems usually distribute information. Global information (coming from a Grid Information System like MDS, BDII, RGMA) presents the overall picture of the grid. Long-term predictions are based on this information. Since global information systems just cache the most important entries of the local systems, in some cases this global information system's historical data is archived for better predictions like in case of MDS4-Archiver (see the monitoring section of CD-JRA-1.2.2 for more details). Local information systems collect live data about the running services. This data is collected from the self-monitoring infrastructure of each service. In case of WSRF ([62]) compliant services the service might offer its self-monitoring information in the singleton resources - these resources usually belong to service interfaces implementing the factory pattern for the creation of other service resources. The dynamicity of the local information makes the predictor engine capable to make short-term predictions.

Finally a self-replicating service skeleton is required in automated deployment systems. This skeleton would be the base for all future systems. And preferably services, which were existing before the introduction of this skeleton, should also be capable to use it without major modifications. The main task of this skeleton is to make replication possible with small impact on the sources of the deployable service. As an example the original service code is used without any changes, however the developer needs to provide the self-monitoring tests to identify erroneous situations while the service is running, and as a response to these situations the skeleton can decide to initiate the replication procedure.

2.3.3 State-of-the-Art

We can categorize Service deployment solutions on their *isolation level*, which defines the level of disturbance caused by deploying a service on a host where other services are already present. The worst case scenario means all the other services are stopped and their states are lost. The best case scenario is when the turnaround time of the other service invocations remain on the same level. The other important disturbance here is the injection of some malicious code by deployment. The isolation level is high when the best case scenario can be provided for other services even when a maliciously prepared code gets activated. There are the two main approaches to tackle the isolation problem. The first follows the fact that a *service container* can offer some isolation, therefore some implementations stay at this isolation level. However service containers do not separate the services entirely - e.g. if the container uses java technologies these services are still running in the same Java VM. The second approach provides isolation with *virtualization* - depending on the level of isolation needed this could range from API virtualization to hardware virtualization. The required level of isolation is linked to the level of trust in the service under deployment.

The service deployment solutions can be also categorized depending on their support of *repositories*. The repositories can be the entities where the trust level is originated from. For example they can enclose security information (e.g. a signature of the service's developer). Or they can enforce different registration policies - like they don't accept services without validation. Similarly to certificate chains another trust chain can be established in case the repositories are used for establishing trust between parties. Repositories can mirror the contents of their trusted peers. Therefore repository users do not need to trust in several different sources, but instead they choose a repository and trust its maintainer.

The next category is the *universality* of the deployment solution (e.g. a deployment service or component). Some service deployment solutions are tuned for a single service. They can support the whole deployment life-cycle from installation towards adaptation and decommission, however to simplify the

problem they only deal with a given service.

Further category can be introduced depending on whether the service developers have to be involved in the preparation of their service for an automated deployment solution or not. Some systems require the service code to be *modified* in order to support them. This can be as simple as implementing an additional interface for the service. However this can also mean the developer has to use specific libraries and solutions to let the further deployments happen.

Finally some deployment solutions assume the *state transfer* should be part of the deployment system. State transfer in SOA would mean that a service can be suspended on a site and resumed on a different one. In this case the whole process depends on the state representation of the service. The deployment system might introduce new interfaces for state transfer, it might require the developer to implement the state transfer mechanisms for its own system, or finally it can also use the WSRF standards [62] which are proposed service interfaces for common state representation and retrieval.

	Isolation	Repository Integration	Universality	Modfication	State Trans
Virtual Worskapces [54]	Virtualization	Partial	Yes	No	No
VMPlants [55]	Virtualization	Yes	Yes	No	No
Hot Deployment serv. [56]	Container	No	Yes	No	No
HAND [63]	Container	No	Yes	No	No
WSPeers [57]	Container	No	Yes	No	No
Dynagrid [64]	Container	No	Yes	Yes	Yes
CDDLM impls. [58]	N/A	No	No	No	No

Table 2.3: Classification of the different deployment solutions

Globus Virtual Workspace Service [54] was not developed for service deployment, however it has been demonstrated (in [65]) that it is also capable for deploying services which are packaged as virtual appliances. These virtual appliances are self contained software packages of the service's code and its support system (like the OS and other dependencies) to be run on a virtualized environment (e.g. on Xen). Therefore the workspace service's isolation level is high since it supports hardware virtualization with Xen, and its generic framework lets the developers support other virtualization solutions easily. Service repositories are not supported on a generic way they are actually handled as a single filesystem. This filesystem can be local or remote to the deployment location. The highly trusted virtual appliances are always stored locally. Because of the nature of the Virtual Appliances the Virtual Workspace Service does not have a limitation of the services supported, since the only requirement for the service that it should be fitted into a single appliance. In the latest versions of the Workspace Service it also supports the cluster on demand concept with which the Service itself might not be in a single VA at all.

VMPlants [55] goes a step further with virtualization and offers faster delivery of the VA image by constructing it on site with the help of deployment directed acyclic graphs. These deployment graphs let the deployment system to build the service from smaller parts coming from a distributed repository called VM Warehouse. Analogous to the Virtual Workspace Service it has two main interfaces, one for managing the virtual machine itself (VMPlant) and one for creating a new one (VMShop). Inside the VMPlant we can find the VM Warehouse which operates a cache for virtual machine construction. Meanwhile the VMShop's main task is to estimate the cost of deployment on a given system and with this information it decides which machine it should initiate the new VMPlant for the requested service. VMPlants defines a framework for the management of Grid virtual machines that includes techniques for representing VM configurations in a flexible manner, for efficient instantiation of VM clones, and for composition of services to support large number of VM "plants". The performance results encourage the use of this technique for on-demand provisioning of Grid VMs, showing that these flexible execution environments can be dynamically cloned often in less than a minute.

Hot Deployment Service [56] uses an entirely different approach for deploying services by introducing new services for the OGSA service containers called ServiceFactory. This interface is a new interface and should not be misinterpreted as the interface to create a WSRF resource - a state representation of a given service. With this interface the system is capable to deploy, undeploy and redeploy services while

the service container is still running. With the HDS the service container needs to be changed (on configuration level) to support the necessary operations for new service registration and class loading. This solution ensures inter-service security by running each service in its own sandbox in order to make sure they don't have direct access to the running code of other services. This means the isolation level is not so high since the execution of a malicious service can interface with the other one by modification of its underlying files or just simply consuming more CPU or network than the others.

HAND (Highly Available Dynamic Deployment Infrastructure) [63] discusses isolation issues which could happen during the activation of a newly deployed service. They extended the Globus Toolkit 4's java web service core container with a thin layer which deals with deployment. Thus this requires the container to have services not coming with the Globus distribution. In Globus's web service core container every service is in a gar file coming with its configuration details, therefore this solution only supports deployment time configuration, reconfiguration requires the undeployment and redeployment of the service. This deployment solution supports several deployment approaches similar to the Virtual Workspace services local and remote deployment locations, and provides two deployment solutions called HAND-C (container level solution which requires the container to be restarted after a service injection), and HAND-S (a service level solution in which all other services are unaffected during deployment).

WSPeer [57] defines a low level message oriented interface which deals with registration, deployment, discovery, invocation. The writers introduce two implementations, one for regular WS based software, and other for P2P Simplified protocol. The deployment in the WSPeer system is simply passing a class file to the WSPeer service which then will register this class as a web service, or in a more complicated case the WSPeer can automatically generate a proxy for a given service which then can be called by the client. This second case is used to interface the WSPeer environment with the Triana workflow environment.

Dynagrid [64] offers an all in one solution which covers all the aspects of WSRF service deployment on some level and even more. They offer a solution for a unified invocation interface (called ServiceDoor), and with the help of the ServiceDoor the authors can even transfer service state. Deployments are done with the help of the dynamic service launcher (DSL) components, which need to be deployed in all the WSRF containers. With DSL they get a unified service interface also, so all the services are hidden behind the invokeMethod function of the DSL service. This is a rather different approach from the previous solutions, because this needs the most changes in the SOA environment but offers not just deployment, but scheduling, migration, and invocation.

2.4 Self-* Services in Mobile and Wireless Environments

Nowadays, the use of portable computers, pocket PC and wireless phone is spreading rapidly. On these mobile devices the users want to perform quite any type of applications, that necessitates to be connected through wireless networks to the outside world, typically to the Internet. Most mobile terminals can use several network interfaces depending on their location: wired network at work, wireless network (WiFi or Bluetooth) outside. All these networks have different characteristics and, for wireless networks, these characteristics may fluctuate significantly. First, the bandwidth is, for most wireless networks, smaller than for wired networks. Moreover, if the mobile moves away from its access point, the bandwidth decreases. Other parameters, like delay, jitter, bit error rate, may vary with the distance between the mobile and the access point, with the number of mobile in the cell thus, their connectivity may be very different over time. Moreover, mobile autonomy due to their batteries and a limited calculating power (related to the available energy). All these limitations need to be taken into account when executing applications on these mobile terminals. To tackle this problem the best thing to do is to give the possibility for software to dynamically adapt to real-time environment characteristics.

2.4.1 Use Cases

Multimedia Applications in Wireless Environments

Multimedia systems are very resource dependent. Using them in a varying wireless environment leads to a need for adaptation in order to cope with, for example, bandwidth variations, screen size and processor power diversity. The transfer and access to multimedia documents that represent big volumes of information may cause problems in mobile and wireless environments.

Self-optimisation: bandwidth adaptation. Codecs and filters can be used to reduce bandwidth usage. For instance, a streaming of MPEG encoded video, which gives good quality but requires a high bandwidth, is not adequate if the available bandwidth decreases. The adaptation consists in choosing another codec (as H263) or another bit rate by changing the compression ratio or the video resolution.

Self-healing: error rate adaptation. The problem with error rate is that some codecs such as MPEG and H263, use temporal redundancy. If a frame contains errors, next frames, which are deduced from this frame, also contain these errors and can add some others. That is why, when the error rate exceeds a predefined value, one must not use that kind of codec and switch to an M-JPEG kind of codec to receive a fresh, complete frame each time. But in that case, the required bandwidth will considerably grow. This means that there is no ”optimal solution”, self-adaptation is needed to dynamically improve the quality of the frames.

Self-configuration: Processor usage adaptation. The processor usage is an important parameter when using codecs such as MPEG4 for video streaming because they require a huge amount of processor time for the video stream to be correctly decompressed. Depending on the available processor power, the adaptation will choose between MPEG4 and MPEG1 for instance.

Transactional Systems in Mobile Environments

In a classical transactional client/server application the requests coming from the client are processed on the distant server that holds all the data bases. During the execution of the transaction the parts of the data bases that are accessed are often locked to ensure consistency. In the case of a mobile user, using a wireless network to be connected to the server, this way of executing the transactions may cause problems. If the quality of the wireless network is poor the response time will increase significantly. Moreover mobile users will not be the only ones to suffer from the consequences of using a wireless network. Indeed fixed users will be blocked by the locks put by transactions initiated from mobile sites. The lower the quality of the wireless network transmission is, the longer are the mobile transactions and thus the more degraded is the quality of service offered to fixed and mobile users.

One solution is to provide a transactional system which can take into account the characteristics of mobile clients [66] [67]. Dynamic adaptation techniques allow to use already existing transactional systems while enabling for instance to adapt transactions processing depending on the wireless network quality of communication. A very simple adaptation strategy could be the following: when the quality of the network is good, transactions are executed on the server, whereas when it is weak, transactions are executed on the mobile host. Executing transactions on the server has the advantage of working on up-to-date data and to make modifications immediately visible to the other users. Executing transactions on the mobile host has the advantage of making the transactions initiated by the mobile user faster, and always possible even in case of disconnection. It also avoids to bother the work of fixed users. However this solution for mobile users necessitates that at least some parts of the data bases reside on the mobile host: data should have been pre-loaded when the connection was good. It has also the disadvantage of working with possible outdated data ; also it delays the knowledge of the modifications made locally by the other users. Some inconsistency could happen, that necessitates reconciliation protocols or rollbacks.

Thats why it is not desirable to always execute transactions on the mobile host and to dynamically change the mode of execution.

2.4.2 State-of-the-art

Adaptation for Multimedia Applications

The need of adaptation is especially important for distributed multimedia applications, which are characterized by an intensive use of host and network resources, and require time bounded processing and communications. Furthermore, multimedia applications need to change their behavior at runtime to fit the conditions of their environment at any time. Providing an efficient access to multimedia services requires that multimedia data be adapted according to each terminal capacity.

A survey and a comparison of existing approaches can be found in [68]. These approaches are broadly classified into: sender-based, receiver-based and network-based adaptations. In the sender-based approach, the sender regulates its transmission according to network conditions [69]. The adaptation mechanism consists in tuning key encoding properties (quality factor, frame rate, etc.) in order to reach the desired transmission rate. The second adaptation approach leaves adaptation decisions to receivers. The suppliers of multimedia content provide several versions or multiple cumulative layers [70] corresponding to different levels of QoS. The receiver can then select the content that agrees with its capabilities. The network-based (or proxy-based) approach consists in integrating media adaptation at intermediate nodes in order to adapt the multimedia content according to network, hardware or software capacities [71]. In [72], the authors address the dynamic configuration and the reconfiguration at runtime and describe a framework for network-based adaptations in multimedia streaming applications. This framework relies on a language, named APSL (Adaptation Proxy Specification Language), allowing the specification of network-based adaptation processes and their run-time behavior. The translation of this language is based on a component-based platform offering basic multimedia-related functionalities.

Chisel [73] is an open framework for dynamic adaptation of services using reflection in a policy-driven, context-aware manner. The system is based on decomposing the particular aspects of a service object that do not provide its core functionality, into multiple possible behaviors. As the execution environment, user context and application context change, the service object will be adapted to use different behaviors, driven by a human-readable declarative adaptation policy script. This framework has been used to provide a dynamically adaptive middleware for mobile computing. It allows users and applications to make mobile-aware dynamic changes to the behavior of various services of the middleware, and allow the addition of new unanticipated behaviors at run-time, without changing or stopping the middleware or an application that may be using it.

The authors of [74] present a framework to model video entities, resources, the required Quality of Service and the adaptation which is needed to link them. A proxy placed between the server and the client implements the adaptation mechanism. An example for adapting MPEG-4 video to dynamic bit rates is shown.

In [75], the authors compare the effects of two adaptations based respectively on quantization and frame rate for streamed video.

In [76, 77], the authors describe a formal approach to context-aware ubiquitous computing: they offer a context-aware action systems framework, which enables the correct-by-construction approach.

Adaptation for Resource Management for Data Replication

Some works on replication consider some environment parameters that are related to the system performance which may indeed be influenced by the change in mobile environment resources. For example, the bandwidth degradation and the change in user location can considerably increase the response time and the network overhead. Among these studies are [78, 79] that take into account, at replica placement time, network topology information and user location in order to reduce the client access latency and the

network overhead. [80] propose a dynamic replication scheme which is based on users' mobility schedules, access behavior and read/write patterns. So, the application designer defines a set of constraints that have to be satisfied in order to create data replicas or perform any operation on these replicas such as read/write operations and consistency. [81] propose a generalized framework for replica hosting system that automatically adapts the replication scheme to variations in some parameters which are related to the system performance and are classified as controllable and uncontrollable parameters. Controllable parameters are those whose value can be controlled by the system e.g, the number of replicas and their location. Uncontrollable parameters are those which cannot be directly controlled by the system like client request rates and update rates for Web documents. [82] propose replica creation in mobile devices to allow the operation in disconnected mode and/or to avoid the use of weak bandwidth links.

However, these works do not take into account the nature of data. Indeed, according to its nature, data may need a set of resources such as network bandwidth and storage space in order to be replicated. Other environment information like node security, device type and user activity may also be essential for replicating or performing any operation on this data. For example, for a banking application, data is confidential. That is, to create and place this data, we need a high security level for nodes and links. The change in some of this information might not influence the performance of the system and the previous works cannot therefore provide a satisfying solution to replication.

Chapter 3

Service Registration, Discovery, Binding and Invocation

Software systems built on top of Service-Oriented Architectures (SOA) use a triangle of the three operations ”publish”, ”find” and ”bind” in order to decouple the participants in the system. These operations are generally the interactions of three distinct roles: service consumers are interested in certain capabilities, and need services that satisfy these needs; service providers provide certain capabilities, and need clients that are interested in them; the service registry is the broker that brings together consumers and providers. Basically, all these roles are technically connected through a SOA middleware, that allows for service discovery, selection, binding, and invocation. Additionally, service middleware often addresses messaging challenges such as message routing and transformation. Additionally, a number of end-to-end issues, such as SOA governance, SLA accordance and service version management, need to be dealt with by the SOA middleware.

Publishing and finding services are essentially registry issues - these operations particularly demand for a highly available, possibly federated and active, service registry, and an expressive interface description and querying language. The third operation, bind, is independent from the registry. Dynamically binding and invoking services puts requirements on the service client, which has to be able to connect to any service that he has detected during the find step. This problem is exacerbated by the fact that different services will, even if they provide the same basic functionality, usually differ in the functional interface they expose, in the Quality of Service level they provide, and in the protocol (that is, in the ordering of invocations) that they expect. Service clients in a SOA need to be able to mediate between these heterogeneities at run-time.

3.1 SOA Registries and Discovery, Web Service Registry and Discovery

In this section a general overview of Service Registries and Service Discovery mechanisms will be presented. It will be described a significant and novel scenario in which registries and discovery issues comes into play. Moreover, it will be presented a classification of the state-of-the-art registry and discovery techniques according to various approaches.

3.1.1 Use Cases

As important european project shown (i.e. NextGRID [17]), the use of Service Level Agreements on Grid and SOA platforms is crucial [83]. Laforenza et al. [84] present *SPRanker*: a service discovery tool that is able to retrieve providers from partially specified service descriptions based on Service Level Agreements. It ranks providers found on the basis of an Information Retrieval-based score formula that takes into account judgments expressed collaboratively by past service users. The aim is to guide customers in collaboratively choosing the best provider for the service they need. The term collaboratively

means that users, i.e. customers, give feedback about providers performance. Collaborative ranking means to exploit users' feedback to rank providers of services. The best providers are, in turn, chosen among those historically provided the service with the best QoS possible with respect to the application domain. To this extent SPRanker will return a ranked list of service providers in response to a customer Query-SLA containing the specification of the service it is looking for.

Novel approaches in application development exploit the benefit arising from *ubiquitous computing* (pervasive grid) and the usage of *context* information to accomplish their tasks. This new application model strongly requires *resource discovery* mechanisms and *self-** capabilities. In particular, the term "pervasive grid" is referred to a computational grid made up of several heterogeneous devices such as clusters, single PCs, laptops, PDAs, mobile phones, and so forth. On the other hand, *context-awareness* refers to the idea that computers and applications can both sense, and react based on their environment. Devices may have information about the circumstances under which they are able to operate and based on rules, or an intelligent stimulus, react accordingly. Context aware devices may also try to make assumptions about the user's current situation.

As an example, the Italian FIRB In.Sy.Eme (**In**tegrated **Sy**stems for **Eme**rgencies) project aims at developing the runtime support infrastructure, enabling specific *context-aware* applications running on *pervasive grids*. Those specific applications are designed for the management of emergency situations due to natural phenomena. In these situations it should be fundamental to provide a resource discovery mechanism that supports their highly dynamic behavior.

Just to set the ground, let us make a very simple example. Some kind of natural event has brought down network connectivity within the working area of an application's part. It is within this reconfiguration phase of the application that a resource discovery mechanism comes in to play. It should exist a context-aware module which will be responsible for replacing those components that are prevented to work in a particular context, invoking a resource discovery module.

The resource discovery module could provide two querying methods: by Unique Resource Identifier (URI), or by specifying (possibly through some semantic query language) non-functional requirements such as: QoS, performance constraints, etc. By using the first query method, a context-aware module will discover the location of a known module (or a plug-in). The second method can be used, for instance, to search for alternative modules having similar features to those specified by manager's programmer, in case any of the modules (or plug-ins) specified by the programmer is available. Therefore, this mechanism enables the application to dynamically adapt itself in response to unexpected context state changes.

3.1.2 State of the art

Tipically service providers advertise their capabilities to middle agents (e.g. service broker). Discovery mechanisms are essentially a match-making process including the following general steps:

- middle agents store the information service providers want to be published;
- a service requester asks a middle agent whether it knows of service providers best matching requested capabilities;
- the middle agent, in order to reply, tries to match the request against the stored advertisements and returns a subset of stored service providers' advertisements.

A more up-to-date approach, proposed by Booth et al. [85], defines the Web Service Discovery mechanism in a broader sense as "*the act of locating a machine-understandable description of a Web Service that may have been previously unknown and that meets certain functional criteria*". The main obstacle affecting the Web Service Discovery mechanisms is heterogeneity between services, concerning substantially 3 aspects:

- *technological* (different platforms or different data formats);

- *ontological* (domain-specific terms and concepts within services that can differ from one another, especially when developed by different vendors);
- *pragmatic* (different development of domain-specific processes and different conceptions regarding the support of domain-specific tasks).

Hereafter, the state of the art about service discovery is introduced taking inspiration from the classification made by Garofalakis et al. in [86] and Klein and Bernstein [87]. Such a classification aims at highlighting how different work deal with the obstacles introduces above.

Garofalakis et al. classifies the existing discovery approach according to three main dimensions:

- Architectural approaches use novel infrastructural design techniques and paradigms on which the discovery process should be based.
- Behavioral approaches investigate how and when such process has to take place in typical service oriented scenarios.
- Modeling approaches exploit classical results in other computer science domains such as information retrieval and machine learning, trying to apply those principles to Web Service discovery.

In this document, to capture the technological obstacle, we focus on the *architectural dimension* and we distinguish between approaches that consider a *centralized* registry or repository for storing Web service descriptions and approaches that rely on *distributed* architectures.

On the other hand, Klein and Bernstein more focus on the way in which the matchmaking occurs, identifying four main kind of retrieval approaches: keyword-based, concept-based, table-based, and deductive. Keyword-based approaches require that all the documents are associated with keywords which the retrieval algorithms are based on. Concept-based approaches rely on defining an ontology of concepts for classifying documents. Table-based approaches consist of attribute-value pairs that capture service properties. Finally, deductive-based approaches require the description of Web services with formal logics. Retrieval then consists in inferring which services achieve the functionality described in the query. As stated by the authors, the keyword-based approaches are the easiest to be implemented, but they provide lower precision and recall. On the contrary, the deductive-based ones appear to be the most precise, but they involve the use of formal logic: Web service capabilities and user queries are more difficult to define and the computational complexity grows.

In this document, we organize the discussion about the *retrieval mechanisms* according to the following approaches: *keyword-based*, *signature-based*, and *semantic-based*.

Finally, our classification also include approaches that consider the *users' context* or the *quality of service* as part of the request.

Architectural classification

Centralized Broadly speaking, a centralized *registry* or repository is an authoritative, centrally controlled store of information. A lightweight version of a registry is the centralized service of indexes. The index is a compilation or guide to information that exists elsewhere: it is not authoritative and does not centrally control the information that it references. The key difference between the two approaches is not just the difference between a registry itself and an index. Rather, the key difference is one of control: in the registry model, it is the owner of the registry who controls retrieved service descriptions, while in the index model, since anyone can create an index, market trends determine which indexes become popular. Hence, the market heavily affects and controls what and how service descriptions get discovered [85].

Web Service registries are the dominating technological basis for Web Service Discovery mechanisms. In particular, *UDDI* [88] is the leading standard registry in Web Service infrastructure, which allows to publish, discover and consequently share information across different partners. A UDDI registry service is itself a Web Service that manages information about service providers, service implementations, and service metadata. However, UDDI provides limited search facilities, essentially based

on keyword-matching techniques in order to discover suitable Web Service providers. Hence, the available search tools are very simple and do not take into consideration any direct cross-correlations between Web Services and the qualitative characteristics of each Web Service, forcing the user to repeat the search from the beginning using new keywords. Indirect cross-correlations are obtained by classification. UDDI includes a set of taxonomies (e.g., UNSPSC) according to which the published web services can be organized. In this way, two Web services connected to the same element of a taxonomy can be considered related.

Although providing basic support for remote service invocation, UDDI does not support dynamic service invocation within a network of distributed services. *Active UDDI*'s [89] basic approach is an extension of the existing UDDI infrastructure without requiring changes to the data structures or the APIs themselves but using a totally new Web Service that plays the role of a man-in-the-middle. This solution provides a proxy based approach in order to dynamically provide registry updates.

A different use for Web Services discovery is presented in Avila-Rosas et al. [90]. In order to build an open, large-scale and interoperable, multi-agent system in the context of Grid computing, an attempt to integrate agent technologies with Web Services is made. The Grid problem is defined as flexible, secure, coordinated resource sharing, among dynamic collections of individuals, institutions and resources [6]. An extension of UDDI registry is used, with additional information (meta-data) about agents and an ontology-based pattern-matching in order to accommodate the kind of searching that is required to locate an agent service according to the performative it supports. The proposed extension of UDDI contains WSDL descriptions of all agents that have been registered. In this way dynamic discovery and invocation of services by software through common terminology and shared meaning is enabled.

An additional centralized approach is given by the *ebXML Registry and Repository* (ebXMLRR) [91]. Focus of ebXMLRR is to support the publication and the advertisement of Web services, provider and user profiles (ebXML CPP), the agreement between providers and users (ebXML CPA), and the definition of business processes that may take place between user and provider (ebXML BPP). Even if UDDI can host this kind of information, ebXML provides an explicit publication and advertise model to support that especially in case of e-business scenario.

Decentralized Decentralized approaches are related to the distribution and sharing of service information across federated service registries. A typical decentralized approach describes publicly available UDDI nodes connected together to form a service that, while appearing to be virtually a single component, is composed of an arbitrary number of operator nodes. They are called the *UDDI cloud or federation* [92]. An operator node is responsible for the data published at this node: in UDDI terms, it is the custodian of that part of the data. Data consistency issues are resolved by the invocation of data replication procedures, inherent in the UDDI. Re-querying the registry faces invocation failures caused by static service caching.

More elaborated decentralized solutions have also been proposed by Schlosser et al. [93] and Sivashanmugam et al. [94]. Typically these systems are built on Peer-to-Peer (P2P) technologies to publish and search for Web Services descriptions like CA [95], Pastry [96] and Chord [97]. A P2P solution is also proposed in Schmidt and Parashar [98]: they present P2P indexing system and associated P2P storage that supports large-scale, decentralized, real-time search capabilities. In Li et al. [99] an architecture called *PWSD* (P2P-based Web service Discovery) is presented. The authors use a Chord P2P protocol as overlay, consisting of Service Peers (SP). Each SP is mapped to several Logical Machines (different machines corresponding to the same hardware). Each Logical Machine maintains the necessary interfaces to map and search Web Services in the P2P network. Service Descriptions as well as queries are hashed and routed in the Chord network. The *Speed-R* system [100], is a Web Service storage and retrieval system that uses ontologies and a P2P infrastructure. Some nodes in the P2P subsystem are assigned registries, which in turn are partitioned according to their specific domain.

Distributed approaches also include agent based solutions. Among the others, Montebello and Abela [101] introduce an environment called *DASD* (DAML Agents for Service Discovery) where Web

Service requesters and providers can discover each other with the intermediary action of a matchmaking service. Distant ancestors of the distributed lookup registries are the Whois++ and rWhois [102] look-up protocols. Both protocols provide online look-up of individuals, network organizations, key host machines etc. Their key attribute is their hierarchical and distributed architecture, in a similar vein but different context with the contemporary decentralized lookup protocols.

Finally, *DIRE* (DIstributed REgistry, [103]) provides a common service model for heterogeneous registries and makes them communicate through a P/S middleware. DIRE is in line with those approaches that tend to unify the service model (e.g., JAXR [104] and USQL [105]).

Information Retrieval mechanisms

Keyword-based approaches Queries in keyword-based approaches are a set of words that are searched in the available Web service descriptions. A Web service is returned, i.e. it is considered relevant, if such words are included in its description, otherwise it is considered not relevant. Information Retrieval literature [106] has lots of approaches for performing this kind of search when keywords should be searched in a set of documents. Since Web service descriptions are documents, some of these approaches are adopted for Web service discovery. Ratnasamy et al. [95] represent service descriptions as document vectors. A description text, thus, corresponds to a vector in the vector space spanned by all the terms used in all Service description texts and also applying the Latent Semantic Indexing (LSI) [107] for including in each vector the more significant semantic concepts. When applying LSI to the discovery of Web Service the authors observed that description vectors resulting from the transformation of the original matrix, were mapped more closely to the vector space representation of the query, than the respective representations of plain keyword-based descriptions.

A Web Service modeled as d-dimensional vectors, can also be thought of as a point in d-dimensions. In that respect a geometric data structure for indexing multidimensional data can be deployed in indexing and querying Web Services. Instead of transferring the problem to high-dimensions, Schmidt and Parashar [98], use a transformation, which injectively maps points in higher dimensions, to numbers. This transformation is called space-filling curve.

Platzer and Dustdar [108] build a Vector Space Search Engine for Web Services. The main idea is to exploit human entered information to search for a given service. Method names, for example, are a good descriptor of what a function is computing (e.g `getMaximumInteger()`, or `searchByString()`).

In [109] and [110], Maiden et al. propose an algorithm for discovering services from requirements expressed using structured natural language developed for the EU-funded SeCSE Integrated Project. The algorithm is based on a pre-processing phase for preparing structured natural language queries and then applies classical Information Retrieval techniques such as query expansion and term disambiguation for retrieving candidate services.

Signature-based approaches Signature-based approaches differ from the keyword-based approaches w.r.t. the way in which the query is defined. Previously, users specify a set of words that the service description documents have to include. Signature matching requires that users specify the interface, or even only a part of it, that the returned Web services have to support. Approaches of this kind rely on the evaluation of similarity between the requested signature and the available ones.

Dong et al. [111] introduce Woogle that receives as input the required operation and looks for all the services that include an operation similar to the requested one. This approach runs a clustering algorithm for identifying the relationships among the terms adopted in all the published services.

Considering the whole web service description, and not only a single operation as a query, the software component retrieval literature has inspired lot of existing work. Signature matching [112] and specification matching [113] algorithms proposed by Zaremski and Wing can be considered foundational. In particular, signature matching considers two levels of similarity introducing the *exact* and *relaxed* signature matching.

Stroulia and Wang, in [114], exploit the description fields as well adopting Information Retrieval techniques (e.g., tf-idf [106]) combined with Wordnet, they are able to increase the precision of the retrieval mechanisms.

In Bianchini et al. [115] the approach also considers the structure, in terms of number of operations and parameters as well as the data type of the messages, of the WSDL and not only the terms included in the description.

Crasso et al. [116] introduce *WSQBE*, a search method on WSDL interfaces that aims at assisting service discoverers by generating a short list of candidate services and easing query specification. In contrast with previous approaches the authors show a new space reduction mechanism.

Laforenza et al. [84] present *SPRanker*: a service discovery tool that is able to retrieve providers from partially specified service descriptions based on Service Level Agreements. It ranks providers found on the basis of an Information Retrieval-based score formula that takes into account judgments expressed collaboratively by past service users. The aim is to guide customers in collaboratively choosing the best provider for the service they need. The term collaboratively means that users, i.e. customers, give feedback about providers performance. Collaborative ranking means to exploit users' feedback to rank providers of services. The best providers are, in turn, chosen among those historically provided the service with the best QoS possible with respect to the application domain. To this extent SPRanker will return a ranked list of service providers in response to a customer Query-SLA containing the specification of the service it is looking for.

Finally, Zisman et al. [117] present a Service Discovery Framework based on linear composition of service operations. When no single operation can be identified, more than one web service operation can be combined to implement a new system functionality. The discovery process is based on best matches between the input and output parameters of service operation request and service operations in the composition. These best matches are calculated based on a graph-matching algorithm that verifies graph-subgraph isomorphism between two graphs representing the data types of the input and output parameters of the service operation request and service operations in the composition. Similarly, Budak Arpinar [118] study the relationships between pre- and post-conditions to figure out which is the goal of the Web service and to provide a tool for automatic composition.

Web service matchmaking can also consider the component behavior. Mecella et al. [119] compare the external behavior of Web services, i.e., their conversations. Considering the conversation as a state machine, their approach compares these state machines in order to evaluate their compatibility. About the similarity of the Web service signatures, Mecella et al. do not deal with names and types compatibility assuming the presence of a domain expert able to manually discover possible similarities.

Semantic-based approaches Semantic-based approaches include all the work that exploit the Semantic Web mechanisms for comparing the service signatures: ontologies are used to improve the semantic of the Web service descriptions (e.g., SAWSDL [120]), or the Web service descriptions are organized in ontologies (e.g., OWL-S [121], WSMO [122], SAWSDL [120]).

Sycara et al. [123] provide the starting point for many of the existing work. They proposes a language, LARKS, and specifically designed for agent advertisement. The matching process is carried out through five progressive stages, going from classical IR analysis of text to semantic match via θ-subsumption. The notion, inspired by Software Engineering, of plug-in match is introduced to overcome in some way the limitations of a matching approach based on exact match. No ranking is presented but for what is called a relaxed match, which basically reverts again to a IR free-text similarity measure.

The approach of Paolucci et al. in [124] uses DAML-S (now OWL-S) to describe user needs and service offers and follows a similar approach. The main rational behind its matching algorithm is that an advertisement matches a request when all outputs of the request are matched by the outputs of the advertisement, and all the inputs of the advertisement are matched by the inputs of the request. The matching algorithm ranks the advertisements that match a request into the following four categories with decreasing order of significance: *exact*, *plug in*, *subsume*, *fail*. *Exact* matches indicate perfect matches.

Plug in matches are offers that provide a more general output than the user demand. *Subsume* matches are offers whose output is more specific than that needed.

Even Li and Horrocks [125] use DAML-S for describing requests and advertisements. In addition they use Racer [126] for computing a Service Profile hierarchy for all advertised services. It produces four categories of matches in a similar way to [124]. However, its difference with algorithm described in [124] is that it compares the whole DAML-S Service Profiles and not the inputs and outputs.

Moreau et al. [127] incorporate a RDF triple store in a service registry in order to integrate the major service descriptions (WSDL, UDDI, and DAML-S) and other task- or even user-specific metadata that can be used for service discovery. Thus it can be used both for syntactic and semantic discovery of all Web Services independent of the format they were stored. This approach also allows the use of metadata in order to characterize entities such as web services (QoS metadata for instance) or in order to provide semantics to WSDL input and output messages.

The approach of adding DAML extensions to UDDI descriptions is also adopted by Sivashanmugam et al. [100]. Instead of providing a separate layer, they perform a simpler construct by enabling both WSDL descriptions and UDDI registrations contain semantic information. This information is simply a mapping between WSDL entries and DAML+OIL ontologies. Considering UDDI, different tModels are provided to represent functionality, input, output, etc. A matching procedure that uses templates and exploits semantic descriptions to provide semantic matching is performed.

Similarly, the approach described in Akkiraju et al. [128] uses DAML-OIL ontologies created by service providers or selected from ontology repositories in order to semantically annotate services described with the DAML-S web service description language. Then the service advertisements are published in a UDDI registry according to the mapping defined in [129].

The semantic matching engine enhances the one described in [124] by offering a flexible composition mechanism when no direct matches are found. Simple service composition is achieved by employing a backward chaining algorithm. When the service annotation relies on SAWSDL, Syeda-Mahmood et al. [130] propose a suitable matchmaking algorithm inspired by [128]. The work demonstrates the effectiveness of using a domain-specific ontology: the precision of their retrieval mechanism increases when a domain-specific algorithm is involved.

In the approach described by Dogac et al. [131], ebXML registries are enriched through OWL ontologies to describe Web Service semantics. These stored semantics can then be queried through standardized queries by using the ebXML query facility. The authors also indicate the reasons why ebXML registries should be used instead of UDDI registries for incorporating and using semantics.

Avila-Rosas et al. [90] also perform some kind of semantic matching but in a different context. They describe agents performing Grid computations, as web services. Hence, they transform agent ontologies into XML and semantic matching is performed by validating structurally-expressed queries against agent description schemas. In the approach of [101] a combination of semantic annotation of web services and agent-based publishing and discovery is followed.

In Yu et al. [132], a user's task requirement is turned into formal request with the help of domain ontologies, NLP and semantic web techniques.This is important as a user should not be required to know DAML-S or other service description languages in order to describe his needs. Then the formal request is passed into three different matching algorithms: the first is used in order to reassure the right service category; the second is clearly a capability matching algorithm; the third is a constraint based algorithm as capabilities are followed by restrictions. The score from each algorithm is summed for each advertisement-request pair and the advertisement with the highest score is selected. The user intentions sometimes are translated to a complex WS, which is created by executing the matchmaking engine for every task. If the created complex WS is totally different than all the other stored in the repository, then a new task flow can be created, registered and stored according to some criteria.

In *PILLAR* [133], the semantic discovery process filter by user domain so that services operating in different domains will be discarded. Then it filters by supporting roles so that services that do not support the user role will be discarded. Then it filters by G^+ (a goal model proposed in the same paper) so that

services that have different goals and contexts will be discarded Finally it filters by technical context so that services that do not match the context requirement will be discarded.

In Kawamura et al. [134], services are described using the Web Service Semantic Profile (WSSP), a way to encode semantic information in WSDL that is inspired by the DAML-S Service Profile. Input and output messages of a WSDL file are associated with domain ontology concepts. In addition, constraints are added to inputs and outputs of a WS by a URI of a fact or rule written in a RDF RuleML file. The proposed matchmaker identifies three levels of matching: exact, relaxed, and plug-in

Di Noia et al, in [135], introduce a matchmaking system which takes as inputs a Knowledge Representation System Specification (KRSS) string describing the demand/supply and a URI referencing the proper ontology. It is assumed that both service providers and requesters describe offers and needs that apply in the same domain of knowledge in a free-text style. Then the system applies NLP techniques to parse user input and in concert with a domain ontology produces a KRSS representation. The matchmaking system is based on the NeoClassic reasoner. Two algorithms, based on a modification of the original CLASSIC structural subsumption algorithm [136], were devised to classify and rank matches.

In Benatallah et al. [137], Web services are described by a restricted subset of DAML-S, in which the property of *structural subsumption* applies. If this property applies in a DL, then the difference operator is semantically unique. Matchmaking can be defined as the procedure in which, after taking as input one request Q and one restricted DAML-S ontology T, we can calculate the best combination of web service descriptions of T that satisfies as much as it can the output of Q and that demands as little as possible from the input that is not defined/determined by Q. These combinations of WS descriptions are called best profile covers of Q using T. For this reason, the matchmaking problem is similar with the best covering problem, which is NP-Hard.

The approach proposed in Balke and Wagner [138] focuses on personalization techniques for the cooperative discovery of user-centered Web Services using an ontology-driven approach. In this approach, it is shown that during the discovery phase the user can be offered an improved quality of provisioning by successively executing/answering service requests that are relaxed along the given service ontology and the user's personal preferences. Moreover, not only user preferences and various conceptions towards relaxation are considered but also generic usage patterns, which are used to describe varying instances of user's preferences and conceptions (relations) with the typical usages of a service.

The same authors, in [139], deal with WS Selection. It is stated that user-service interaction involves three steps: discovery, selection, and execution. For this reason, and for finding useful means of personalization for each step, a user's profile is divided into three parts: a) decomposing goals by typical usage patterns anticipated by the service provider; b) preferred service parameters help in choosing same-task services; c) preferred characteristics of objects or information a service claims to provide. The proposed algorithm for cooperative selection of appropriate services does the following: 1) It expands service request by user-specific demands and wishes; 2) it discards services not matching a certain profile and sorts services according to user provided strategies; 3) it uses techniques from cooperative answering to get a better service selection, including user's long term profiles, without risking to produce empty result sets.

Interesting implementation frameworks for semantic matching are proposed by Hu [140] and by Overhage and Thomas [141]. The first paper provides a framework that uses ontologies to discover ("bind" in that case) the web services that best match a specific operation domain (desired set of operations). Overhage in [141] proposes the implementation of semantic descriptions as an extension of the UDDI protocol, termed as *E-UDDI*. E-UDDI introduces "blue pages", sections that contain semantic descriptions of Web Services; the latter are implemented in DAML-S. The model described in [141], also provides extensions to the green page section of the UDDI, by adding the capability to define constraints in the Web Service execution sequence. However E-UDDI seems to be more a vision than an implementable system.

In [142] Ma et al. proposed CPLSA, a novel approach for efficiently finding Web Services using classical machine learning techniques in order to capture semantic information. Given a query according to the user needs, this method firstly retrieves a set of candidate Web Services using keyword-based syn-

tactic analysis through standard discovery mechanisms based on WSDL and UDDI. This set of candidate services is filtered via a clustering algorithm removing those Web Services whose contents are not compatible with the user's query. As a next step, Probabilistic Latent Semantic Analysis (PLSA) is applied to this new working set, which is further clustered in semantically related groups. Semantic concepts hidden in user's query and services' advertisements are extracted and the matchmaking process occurs at an advanced concept level.

Dietze et al. [143] proposed a fuzzy, similarity-based matchmaking between "real world" context and predefined semantic web services capability descriptions that enable the description of situation characteristics as members in geometrical vector spaces. By extending Semantic Web Services, it is possible to discover the most suitable resource, whether data or services, for a given situation based on semantic similarity, computed in terms of the Euclidean distance between a real world situation and predefined resource descriptions as part of Semantic Web Service capability representations.

Context-based approaches The importance of context in runtime service discovery has been increasingly recognized and several approaches have been developed to support context awareness in this form of discovery process [144, 145, 146, 147, 148].

Work analyzed in Maamar et al. [149] describes a framework that combines Agent-based computing and Context-aware computing in order to offer better user-tailored services in pervasive environments. Context is assumed/modelled as user context, computing, time and context history. This type of context is gathered and stored by agents/sensors in a RDF-based form called CSCP, which provides features like adding condition and priorities to attributes and allowance of multiple conditional RDF-based statements for an attribute. CSCP is a complete standard for expressing user preferences. Context-aware discovery is performed by the Context Manager Agent. This agent takes a user query and extracts the user's profile. Then, based on a search radius, it searches for registries, it issues the user query to these registries and collects back the results. The result set gets filtered according to user context and the web service description constraints. If the results are not enough, the search radius is increased.

In [150] Spanoudakis et al. describe a platform that supports context aware runtime service discovery based on structural and behavioral service models as well as complex context related service discovery conditions which are specified using a newly introduced query language, called *Context Aware Service Query Language* (CA-SQL).

In [151] Zisman et al. describe a framework for runtime service discovery in both pull and proactive push modes that supports quality and context constraints. The runtime push service discovery process supports the identification of services in a proactive way due to (i) changes in the functional, quality or context characteristics of a service, (ii) availability of a new service that can provide the functional as well as quality characteristics of a system in a better way.

Quality-based approaches

QoS in Web Service provisioning is a hot research topic today. The S-Cube workpackage JRA-1.3 has already deeply described the efforts which have been spent in identifying, and defining the QoS attributes in its first deliverable (PO-JRA-1.3.1). Moreover, interesting works concerning these topics have been proposed in [152, 153, 154]. There is some work aimed to improve QoS using either service replication [153], load distribution [155], or server redirection [156]. Other efforts have been put in measuring QoS attributes like *availability* [157]. QoS is extended with the concepts of Quality of Experience (QoE) and Quality of Business (QoB) by van Moorsel in [158]. The author claim these two metrics to be important to quantify customer satisfaction.

In the literature service discovery using QoS parameters have been only partially examined.

In [152] a QoS approach to service discovery is presented for SOAs using QoS to initiate service negotiation. This approach introduces in the negotiation phase a so called "*Web Service QoS certifier*" that, by means of a QoS enhanced UDDI registry, allows the search of services with a requested QoS

level. Even though the approach shown in [152] gives the possibility to enrich searches through QoS information, the results of a search does not take into account historical transaction data.

The work described in Deora et al. [159] refers to calculation of values of those QoS attributes of a WS that it is not realistic to be provided/measured by service providers or third-party software modules. This work suggests a mechanism based on user ratings and expectations for the calculation of the value of a QoS attribute. It proposes that we should take into account only the ratings of users that have the same expectation for the value of a QoS attribute as with the one of the requesting user. This expectation does not only depend on the advertised value of a QoS attribute but also on other (contextual) factors like previous user experience from the service, the service usage cost, and recommendations from friends. Unfortunately, this work presents two major shortcomings: a) all users submitting ratings are trusted; b) user expectation is represented by a single value.

QML (QoS Modelling Language) [160] is another research effort for the specification of QoS. Its main modeling element is a Service Profile that represents a list of QoS contracts/requirements in the form of constraints on quality dimensions. Matchmaking is based on the concept of *contract conformance*: One (QoS) Service Profile P is matched with one client profile Q if all contracts of P conform to all the contracts of Q. Contract conformance is translated into constraint conformance.

Fugini et al, in [161], a hierarchical quality model is presented to capture the relationship between quality dimensions. Relying on the Analytic Hierarchical Process (AHP), users are helped in expressing their preferences and a matchmaking algorithm is introduced to state the fulfillment of the users' requirements with respect to the service offering. An extension of this work, in Cappiello et al. [162] also considers a negotiation step for automatically generating SLAs.

In Zhou et al. [163], DAML-S Web Service description language is extended to include a QoS specification ontology. In addition to the DAML-S extension, a novel QoS matchmaking algorithm is proposed, which is performed after functional matchmaking, and separates the result-set in five categories. This QoS matchmaking algorithm is based on the concept of QoS profile compatibility that states the following: "Two QoS ontology descriptions, say C_1 and C_2, are compatible if and only if their intersection is satisfiable: $compatible(C_1, C_2) \Leftrightarrow \neg (C_1 \cap C_2) \sqsubseteq \emptyset$". The main problem of this approach is that it uses OWL's cardinality constraints in a faulty way.

The research effort described in Ruiz Cortés et al. [164] uses a symmetric and syntactic QoS model expressing mathematical constraints for QoS metrics and user preferences. Before matchmaking, a QoS specification is transformed to a CSP which is checked for consistency i.e. if there is an assignment of values to metrics (a solution) such that all the constraints are satisfied. Matchmaking is performed according to the concept of conformance, which is used for checking out if every solution to the CSP of the offer is also a solution to the CSP of the demand. After matchmaking, two main result-sets are produced: matched offers, not (completely) matched offers. Unfortunately, the consistency and conformance of CSPs does not always lead to polynomial computation of solutions especially if there are non-linear expressions at the QoS constraints. This is a characteristic of the class of CSP problems. Concerning Web Service Selection, the (QoS) score of a Web Service advertisement is calculated as a weighted sum of the weight of each metric multiplied with its utility assessment value, where the assignment of values to metrics is chosen so that the sum is the minimum. That is Web Service Selection is expressed as a Constraint Satisfaction Optimization Problem (CSOP) [165].

Kritikos and Plexousakis propose OWL-Q [166], an ontological approach for QoS-based WS description. In addition, they have devised a QoS metric matching algorithm [166] for aligning OWL-Q specifications, increasing in this way matchmaking accuracy. Finally, they translate OWL-Q to Mixed-Integer Programs (MIPs) and propose a novel QoS-based matchmaking algorithm [167] that provides: a) results even for over-constrained QoS demands and b) an advanced categorization of results with high precision and recall.

Sahai et al [168] present an interesting architecture aiming to specify and monitor guarantees in commercial grids through SLA. Sahai *et al.* use an architecture that automatically monitors data and triggers evaluations of the registered SLAs. Furthermore, authors define a unambiguous language for

formalizing SLAs. A distributed and automated SLA monitoring engine aimed to obtain measurements in a cross-enterprise scenario is presented in a next work in Sahai et al. [169]. SLA monitoring is difficult to automate and it would need precise and unambiguous specification and a customizable engine that collects the right measurements, models the data and evaluates the SLA at certain times or when certain events happen. In a cross-enterprise scenario like web services it is important to obtain measurements at multiple sites and to guarantee SLAs on them.

In [170] Bhoj *et al.* present an architecture that uses contracts based on SLAs to share selective management information across administrative boundaries. They also describe a prototype of this architecture used to automatically measure, monitor and verify SLAs for internet services.

Czerwinski et al., in [171], present an architecture of a secure Service Discovery Service (*SDS*). Service Providers can use the SDS to advertise complex descriptions of available or already running services, while customers use the SDS to compose complex queries for locating these services. The SDS provides a highly-available, fault tolerant, incrementally scalable service for locating services in the wide area. Security in this paper have a key role. SDS can, where necessary, use communication both encrypted and authenticated.

In Al-Ali et al. [172] authors present an extension of the service abstraction in the Open Grid Service Architecture for Quality of Service properties. QoS enforcement often requires mechanisms such as advance or on-demand reservation of resources, varying in type and implementation, and independently controlled and monitored. The focus of the paper is on the application layer, whereby a given service may indicate the QoS properties it can offers, or where a service may search for other services based on particular QoS properties.

In Miles et al. [173] authors propose that Grid- or Web-Services have to be located via personalized, semantic-rich discovery processes. They argue that such processes must rely on the storage of arbitrary service metadata originating both service providers, and service users. Examples of such metadata are: reliability metrics, quality of service data and semantic service description markup. The paper presents UDDI-mT; an extension to the standard UDDI service directory approach that supports the storage of such metadata via a tunneling technique that ties the metadata store to the original UDDI directory. They also discuss the use of a rich, graph-based RDF query language for syntactic queries on this data. Finally, they analyze the performance of each of these contributions in their implementation.

In Xu et al. [174] authors study the issue of wide-area service discovery in service grids. They start with an existing basic wide-area service discovery framework. The key technique they introduce includes: (i) the addition of QoS feedback capability to clients; and (ii) the caching and propagation of discovery results with QoS feedbacks in the discovery server hierarchy. With this techniques, the enhanced service discovery framework will be faster in finding qualified service providers.

3.1.3 Comparison

Table 3.1.3 summarizes the Web service discovery approaches described in this section. In some cases, the architectural style is not defined (we use the hypen) since the deliverable only deal with the matchmaking algorithm without specifying any specific architecture.

Paper	Centralized	Decentralized	Keyword-based	Signature-based	Semantic-based	QoS	Context
UDDI [88]	Yes	No	Yes	No	No	No	No
Active UDDI [89]	Yes	No	Yes	No	No	No	No
Avila-Rosas et al. [90]	Yes	No	No	No	Yes	No	No
ebXMLRR [91]	Yes	No	Yes	No	No	No	No
Schlosser et al. [93]	No	Yes	Yes	No	No	No	No
Sivashanmugam et al. [94]	No	Yes	No	No	No	No	No
Schmidt and Parashar [98]	No	Yes	Yes	No	No	No	No
Li et al. [99]	No	Yes	Yes	No	No	No	No
Speed-R [100]	No	Yes	No	No	Yes	No	No
DASD [101]	No	Yes	Yes	No	No	No	No
DIRE [103]	No	Yes	Yes	No	No	No	No
Ratnasamy et al. [95]	-	-	Yes	No	No	No	No
Platzer and Dudstar [108]	Yes	No	Yes	No	No	No	No
Dong et al. [111]	Yes	No	No	Yes	No	No	No
Stroulia and Wang [114]	-	-	Yes	Yes	No	No	No
Bianchini et al. [115]	-	-	No	Yes	No	No	No
Bianchini et al. [115]	-	-	No	Yes	No	No	No
WSQBE [116]	Yes	No	No	Yes	No	No	No
SPRanker [84]	Yes	Yes	No	Yes	No	Yes	No
E-UDDI [141]	Yes	No	No	No	Yes	No	No
Zisman et al. [117]	Yes	No	No	Yes	No	No	Yes
Budak Arpinar [118]	-	-	No	Yes	No	No	No
Mecella et al. [119]	-	-	No	Yes	No	No	No

Table 3.1: Comparison of the main contributions to Service Discovery (cont'ed).

Paper	Centralized	Decentralized	Keyword-based	Signature-based	Semantic-based	QoS	Context
Sycara et al. [123]	Yes	No	Yes	No	Yes	No	No
Paolucci et al. [124]	Yes	No	No	Yes	Yes	No	No
Li and Horrocks [125]	Yes	No	No	Yes	Yes	No	No
Moreau et al. [127]	-	-	No	Yes	Yes	No	No
Sivashanmugam et al. [100]	Yes	No	No	Yes	Yes	No	No
Akkiraju et al. [128]	Yes	No	No	Yes	Yes	No	No
Syeda-Mahmood et al. [130]	Yes	No	No	Yes	Yes	No	No
Dogac et al. [131]	Yes	No	No	Yes	Yes	No	No
Yu et al. [132]	-	-	No	No	Yes	No	No
PILLAR [133]	-	-	No	No	Yes	No	No
Kawamura et al. [134]	-	-	No	Yes	Yes	No	No
Di Noia et al. [135]	-	-	No	No	Yes	No	No
Benatallah et al. [137]	-	-	No	Yes	Yes	No	No
Balke and Wagner [138, 139]	-	-	No	No	Yes	No	No
Hu [140]	-	-	No	Yes	Yes	No	No
Overhage and Thomas [141]	Yes	No	No	No	Yes	No	No
Ma et al. [142]	Yes	No	No	Yes	Yes	No	No
Dietze et al. [143]	-	-	No	Yes	Yes	No	No
Maiden et al. [109, 110]	-	-	Yes	No	No	No	No
Maamar et al. [149]	-	-	No	No	No	No	Yes
Spanoudakis et al. [150]	Yes	No	No	No	No	No	Yes
Zisman et al. [151]	-	-	No	No	No	Yes	Yes
Deora et al. [159]	-	-	No	No	No	Yes	No

Table 3.2: Comparison of the main contributions to Service Discovery.(cont'ed)

Paper	Centralized	Decentralized	Keyword-based	Signature-based	Semantic-based	QoS	Context
Fugini et al, in [161]	-	-	No	No	No	Yes	No
Zhou et al. [163]	-	-	No	No	No	Yes	No
Ruiz Cortés et al. [164]	-	-	No	Yes	No	Yes	No
Kritikos and Plexousakis [166]	-	-	No	Yes	No	Yes	No
Sahai et al. [168]	-	-	No	Yes	No	Yes	No
Bhoj et al. [170]	-	-	No	Yes	No	Yes	No
Czerwinski et al. [171]	-	-	No	Yes	No	Yes	No
Al-Ali et al. [172]	-	-	No	Yes	No	Yes	No
UDDI-mt [173]	Yes	No	No	No	No	Yes	No
Xu et al. [174]	No	Yes	No	No	No	Yes	No

Table 3.3: Comparison of the main contributions to Service Discovery.

3.2 Dynamic Binding and Invocation

In order to fully realize the benefits of service-based systems clients need to be decoupled from the service provides they are using. In a nutshell, this means that clients should not be hardwired to specific services, but should be able to roam freely between different internal and external business partners. This is facilitated by dynamic binding and invocation, and increases flexibility, fault-tolerance and reliability of service-based systems.

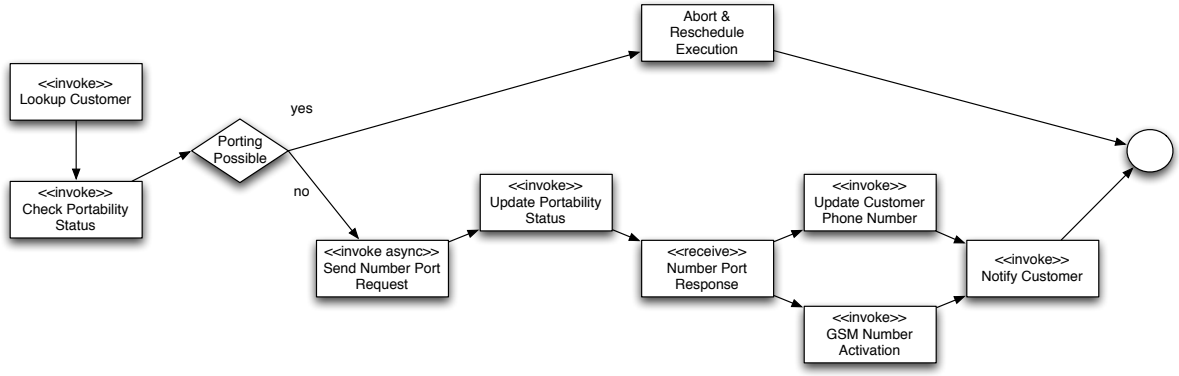

Figure 3.1: Example Process

The simple business process in Figure 3.1 exemplifies the need for dynamic binding. This process is from the mobile phone provider domain, and shows the (simplified) steps necessary in order to port an existing mobile phone number to a different provider. Even though the business process runs locally at a provider's site, some of the activities in the process (e.g., "Send Number Port Request") refer to services of external competitors. Additionally, some of the activities are implemented by a number of different internal services (e.g., "Notify Customer" can be implemented by an e-mail service, a short message service, or a postal service).

Dynamic binding and invocation demands for a client that is flexible enough to be able to connect to different services, possibly provided by different departments or companies, at run-time. Since these services will usually be heterogeneous clients also need to be able to resolve interface and protocol incompatibilities transparently.

3.2.1 State-of-the-Art

[175] states that we currently experience a certain divergence between SOA theory and practice (Figure 3.2). The authors believe that the concepts of "publish-find-bind-execute" are currently "broken": usually, there is no dedicated registry entity available in service-based systems (cp. Section 3.1). Service clients invoke their target services through well-known exact endpoints and interfaces, and are, therefore, statically bound. If the target service fails, or does not deliver a reasonable Quality of Service level the binding cannot be changed easily. Such an architecture does not deliver the SOA promise of flexibility through loose coupling.

As a remedy to these problems, the authors propose VReSCO, an integrated SOA runtime that facilitates loosely coupled services and clients by (1) providing an active, QoS- and composition-aware service registry, and (2) allowing clients to be written against abstract service requirements rather than concrete service instances.

Listing 3.1 exemplifies how a dynamically rebinding client can be written using the VReSCO runtime environment. In this example, the client is not bound to a concrete service instance, but will always be rebound transparently to the service instance best matching his query (QoS.ResponseTime $<=$ 500ms and QoS.Availability $>$ 0.9). For dynamic invocation, VReSCO relies on runtime generation of proxies.

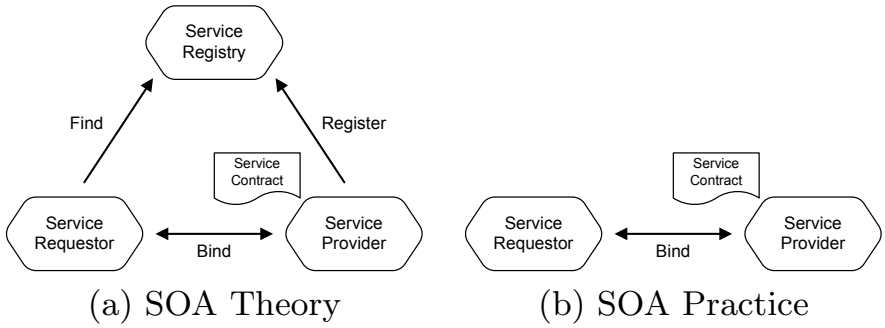

Figure 3.2: SOA Theory vs. Practice

```
1  vresco.registerBindingListener(this);
2  service = vresco.query("MessagingService",
3    "QoS.RersponseTime <= 500ms
4     and QoS.Availability > 0.9",
5    QueryMode.Relaced);
6  vresco.setRebindingStrategy(
7    BindingStrategy.UpdateOnBetterMatch);
8  ServiceProxy proxy =
9    vresco.createServiceProxy( service );
10 proxy.sendMessage("sendSMS");
```

Listing 3.1: Dynamic Binding in VReSCO

[176] presents WS Binder, an earlier approach to dynamic binding developed within the SecSE project. The WS Binder project focuses on QoS- and availability-based dynamic binding in the context of service compositions. The assumption of WS Binder is that service bindings in a BPEL process may have to be exchanged (i.e., rebound) at run-time, because (1) the initial assumptions of a services QoS may turn out to be wrong, (2) a new service is published that better fits the criteria of the selection, or (3) a service may become unavailable. WS Binder allows for three different type of dynamic rebinding: services can be exchanged either before executing a workflow or at run-time of the workflow (i.e., just before the actual service has to be invoked), and the workflow itself may be rebound (i.e., a slice of the workflow may be changed at run-time).

[177] puts forwards some work with similar scope to WS Binder: in this paper the idea of reusing service compositions by applying the same abstract choreographies to different sets of concrete services. This can also be seen as dynamically binding an abstract process to concrete services. These ideas have been incorporated into JOpera, a Swiss-based research platform for service composition. The work identifies a number of different types of binding, namely registration time, composition time, compilation time, deployment time, startup time, invocation time, and failed invocation time binding. JOpera implements these different types of binding using a visual composition language, which is based on the concept of reflection.

So far we have mostly discussed the dynamic binding of services, without explicitly considering the dynamic invocation aspects. Dynamic invocation of Web services has been pioneered by the Web Services Invocation Framework (WSIF), which has been introduced within [178]. WSIF includes a dynamic invocation interface (DII). However, this DII is in general limited to a few pre-defined types (such as string, integer, short ...). Complex, user-defined types need to be mapped to Java types before an invocation, which severely limits the usefulness of WSIF in truly dynamic scenarios.

Recently, new work in the area of dynamic service invocation has been presented by [179]. This work introduces the Daios framework, which aims at decoupling clients from the services they are using by allowing for completely stubless and dynamic Web service invocations using a simple messaging-based interface. Additionally, Daios abstracts away from service internals such as service operations, encoding

styles or physical endpoints. Daios supports the invocation of SOAP-based and RESTful Web services through an uniform interface. The main idea of the framework is to allow users to submit the input that they want to send to a service in a high-level representation (i.e., as "Daios Messages"), which will then be matched using a similarity algorithm against the actual WSDL interface of a service (see Figure 3.3).

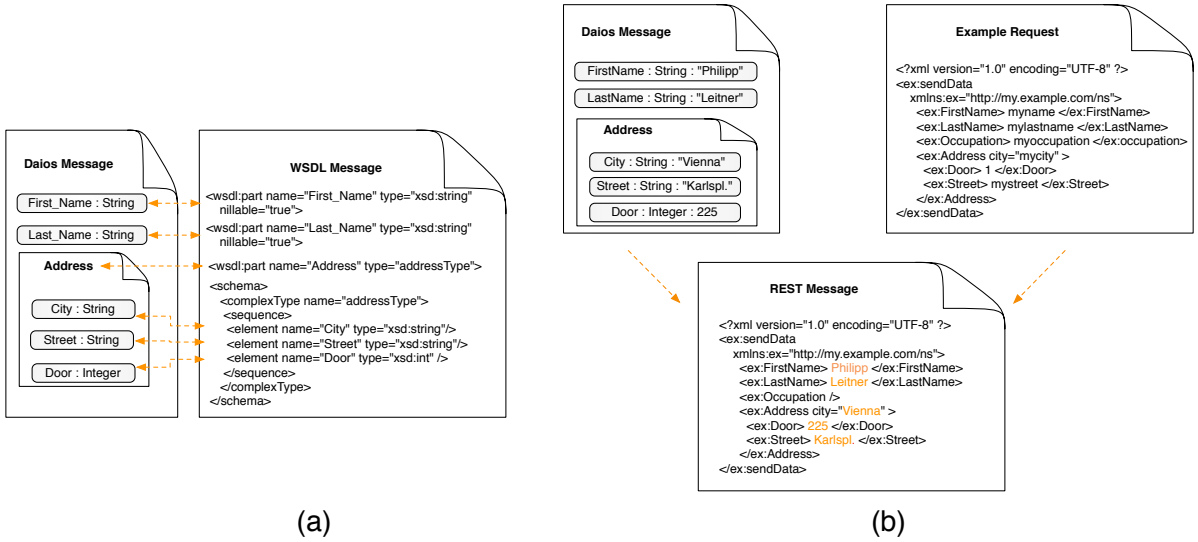

Figure 3.3: Matching Daios inputs to WSDL and REST

That way Daios figures out what operation in the services WSDL interface matches the input of the user best, converts the high-level representation into the respective low-level format (e.g., SOAP using document/wrapped encoding), and dynamically issues the invocation. A respective response from the service is again returned in a high-level representation. Additionally, input may also be transformed into RESTful Web service invocations. Recently, the Daios framework has been integrated into the VReSCO runtime environment (presented above), to replace the proxy-based dynamic invocation facilities.

However, the ability to dynamically bind to and invoke Web services introduces new difficulties. Generally, services provided by different departments or companies will be heterogeneous. Currently, these incompatibilities are resolved using dedicated mediation middleware, such as Web service integration adapters [180]. These adapters need to mediate between different interfaces, i.e., the set of operations supported by services, and business protocols, i.e., the allowed sequences of messages. Increasingly, semantic approaches are used to implement these adapters (see for instance WSMX[181], which is based on WSMO[122]). A full integration of mediation between incompatible services with dynamic binding and invocation is still an unresolved research challenge today.

3.3 Version Management in Service Infrastructures

3.3.1 Use Cases

Software systems in the real world are subject to permanent change. Constantly, new functionality is added to systems, requirements change and software defects and limitations are detected and fixed. In this respect, service-based systems are no different than traditional software systems: single services as well as service-based applications as a whole evolve over time, reviving the need to manage changing service contracts and business protocols. Additionally, new facets are added to the versioning problem in SOA environments. On the one hand, service providers often want to provide several versions in parallel, offering specific variants to some customers or older service versions for legacy applications. On the other hand, some service requesters may want to access different service versions in a uniform manner

or even switch between them at runtime, while others do not want to explicitly deal with different service versions. Therefore, the main issue besides managing multiple versions of a service in the registry is to provide a certain degree of version transparency towards clients. The latter should be able to dynamically invoke different versions at runtime without having to modify their code base.

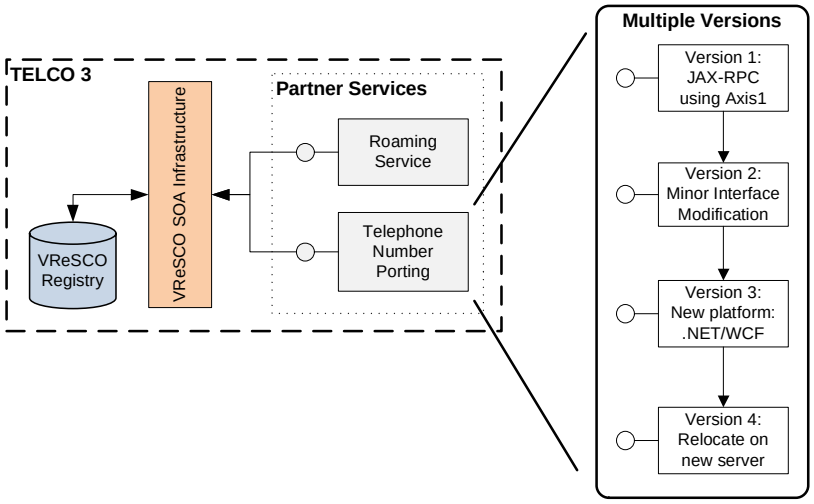

Figure 3.4: Service Versioning

Figure 3.4 presents a possible scenario for version management in service infrastructures: in this example, a certain services (Telephone Number Porting) has been implemented in 4 different versions. The first version of the service has been deployed using JAX-RPC and Apache Axis. Further versions have seen slight interface modifications, and a move towards the Microsoft WCF platform. Finally, the service is relocated to a more powerful machine.

Originally, service registries such as UDDI or the ebXML registry did not deal properly with versioning issues [182]. However, recent work in the area has either extended UDDI and/or ebXML, or proposed new version-aware registries or middleware, allowing for end-to-end version management of service-based systems.

3.3.2 State-of-the-Art

As a first step towards solving the service version management problem, practitioners aimed at developing a set of rules of thumb and best practices for using UDDI as a version-aware service registry [183]. They propose two alternative practices of mapping version information to UDDI: (1) either defining a new interface for each version, and advertising compliance to each of these interfaces, or (2) introduce a version number in the UDDI tModel to identify versions. However, they generally recommend to use the first approach. Both of these approaches have their specific limitations: they both are pure registry-side solutions, that do not take into account the client-side issues of version management; additionally, these approaches are not able to map more complex service evolutions, where branches are created and merged, and business protocols vary during the lifetime of the service.

[184] proposed a different, service provider-side approach to service evolution. They use the Chain of Adapters pattern to manage incremental service version updates (Figure 3.5). They implement a version adapter for each new version of a service, with the responsibility of mediating between the new version and the previous one. Creating new versions involves the implementation of a modified service interface, and an adapter for the previous service version. All versions of a service are kept online for legacy clients. The disadvantage of this approach is the enormous overhead introduced by the Chain

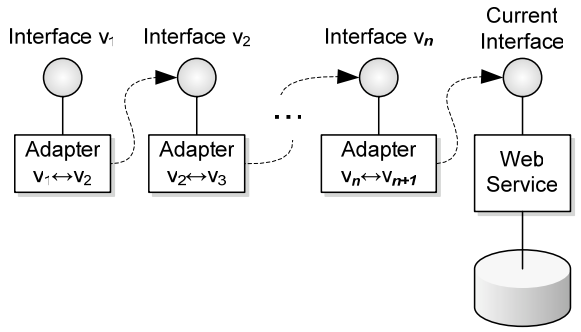

Figure 3.5: Adapter-based Service Evolution Management, from [184]

of Adapters for services with big version histories. Additionally, it might be undesirable for providers to always keep all past versions of a service online. Furthermore, the approach bears the disadvantage that online linear service evolution is considered an evolution scenario where two or more branches of a service evolve in parallel is out of scope of such a solution.

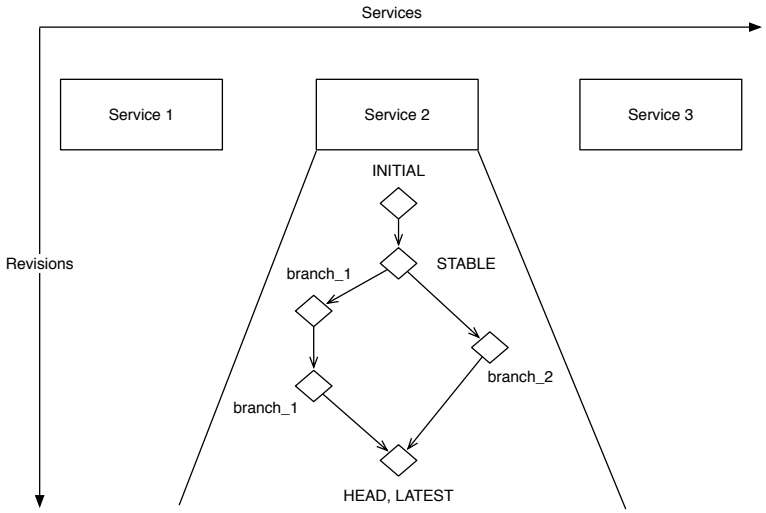

Figure 3.6: Service Version Graphs

This problem has been dealt with in [182]. In this paper, the authors propose to use service version graphs to manage arbitrarily complex evolution scenarios (Figure 3.6). Their approach is implemented in the VReSCO SOA runtime environment presented in [175]. The VReSCO solution outlined here considers version management in an end-to-end way: a version-aware service registry organizes services in service version graphs and categorizes changes, while rebinding proxies are used to dynamically update clients to the service version that is most appropriate for them. Version tags are used to identify specific versions.

[185] considers service evolution on a higher level of abstraction. The contribution of this work is the identification of factors of influence for Web Services (hosting environment, provider, developer and consumer/integrator), and the presentation of a general framework to collect and integrate evolution-relevant service information from various sources into a single data store.

Chapter 4

Future Research Challenges

4.1 Introduction

The discussion of self-* properties and service registry, discovery and binding is concluded by a survey of open issues and potential research directions. This section is not intended to serve as a research roadmap for this S-Cube WP rather, it is a discussion of issues related to the presented state-of-the art: what can and what cannot be solved today. Experience and opinion of experts in this field are gathered here; this section is more like a list of hot topics in research that can be a starting point for envisioning research issues related to services. The research vision for the workpackage is partly based on the outcome of this deliverable.

4.2 Research Challenges related to Self-* Computing

[186] summarizes some of the research challenges of autonomic computing. Self-* services are a specific example of autonomic systems hence these issues largely apply to our research in self-* service infrastructure. As it was discussed, autonomy may appear at any level (any element, component, entity) of computing. With a reference to the categorisation introduced in Section 2, these are grouped as the *level of autonomic elements*, *level of autonomic system* and *human-computer interactions*. We introduce them in generality but with a more specific, in-depth description of those areas that are more tightly related to the current research activities defined in this group.

The *level of autonomic elements* involves any type of computing resource, e.g. storage devices, application servers, load balancer, broker entities, etc. They also can be considered as services in a SOA. On one hand effective interoperability between elements, adaptation of standard interfaces defined by system architectures and holistic, system-level view are largely unsolved. Additionally, research of monitoring techniques, data analysis, event correlation and rule engines are necessary. Beyond these basic technical issues, the vast variety of correlation between events and thus, the complexity of corresponding rules likely require the machine assistance at establishing the rule bases hence, machine learning, automatic discovery of relations, analysis of historical data, modeling and dynamic model refinement, handling uncertainty of open environments are of major challenges.

The level of *level of autonomic elements* roughly corresponds to the self-* infrastructure research group and hence, the following more concrete challenges are distilled. (i) Research on frameworks suitable for both self and controlled adaptation allowing a balance between some level of autonomy of services and some control, and (ii) research on decisions mechanisms and realization of concrete adaptation of services. These mechanisms should also take into account the distributed aspect of SOA.

At the *level of autonomic systems* the main concern is the seamless interactions of autonomic elements. Roughly speaking, a set of autonomic elements is not an autonomic system. Some self-* functionalities involve the cooperation of multiple autonomic elements, e.g. self-deployment or the localisation

phase of self-healing is not possible in an isolated environment. Research issues on one hand involve the problems of autonomic elements: monitoring, analysis, rule and correlation techniques, machine learning, etc. – at a higher level. On the other hand there are issues that are specific to this level, e.g. the scope of problem localisation and remediation that involves compiling a database of problems, symptoms and effects of potential remedy actions. Alternative and novel approaches include reinforcement learning techniques where faults are injected into the system on purpose and various alternative test and repair actions are tried and observed. Further examples of specific problems are self-optimisation and coordination where individual autonomic elements must be controlled so that not the elements but the overall system reaches an optimal working state (this might violate the self-optimising strategies of the elements.)

A potential research challenge in this WP related to the level of *autonomic systems* (and also to autonomic elements) could be (i) investigating higher level, nature inspired models. Many phenomena in nature are self-regulatory, self-evolving towards equilibrium, minimum/maximum, optimum, etc. Appropriate abstraction of these phenomena and application to some aspects of service infrastructure can help with establishing self-* service infrastructures. This area is largely unexplored. (ii) Furthermore, Grid Computing solutions support SOAs that require high performance and distributed computing resources. The presented state-of-the-art sections revealed that self-* properties are missing from existing approaches and there is a definite need for enhanced monitoring, virtualization and brokering with autonomous and adaptive service-based solutions. (iii) Dynamism in SoA can be achieved in two basic ways: adapting available services to the current needs, or deploying more/new services to cope with the requirements of the changing environment. Current deployment solutions are too rigid to support their automation and are focusing on simple use cases involving the developer of the administrator of the services. administrator. Better automation support therefore would require research to support newer use cases identified in section 2.3.

The highest level of abstraction is represented by the *human-computer interactions*. An obvious challenge is developing appropriate interfaces, languages and visualisation tools. A major issue is how the high-level policies and objectives expressed by humans in a natural manner can be represented and transformed into lower-level rules and mechanisms in the autonomic system. Recently utility functions have been introduced that are able to quantitatively describe the state of the system with respect to some objectives; utility functions are in focus of research. This latter field of research in autonomic computing is largely out of scope of this WP.

4.3 Research Challenges related to Service Registry, Discovery, Binding and Invocation

Current research in the area of dynamic Web service binding and invocation mostly considers resolution of service heterogeneity (resolving incompatibilities between services). We refer to the process of incompatibility resolution as service mediation. Generally, we can distinguish the following types of incompatibilities: (1) technical incompatibilities (2) service interface incompatibilities (3) service protocol incompatibilities and (4) semantic mismatches. Interface incompatibilities consider different representations of semantically identical service invocations, while protocol incompatibilities mean differences in the ordering of service invocations in order to reach a certain high-level goal (e.g., in one service a single service invocation may be sufficient to produce a certain output, while in another one 2 operations need to be invoked sequentially to do the same). By definition, protocol incompatibilities can occur only in stateful Web services. Technical incompatibilities can be dealt with by stricter adherence to standards and attract little research attention. Most research work is attracted by incompatibilities on interface and protocol level, with a current trends towards focusing on service protocols. Semantic incompatibilities are considered rarely, since mediation between semantically different services is rarely useful. Currently, most work in the area of service mediation uses Semantic Web Services technologies. How-

ever, real-world Semantic Web Services are sparse. As a future research topic, the community therefore needs to develop concepts and tools to enable mediation between services without relying on semantics technology.

Most of the approaches shown in this document are referred to traditional SOAP-based Web Services. However, the upcoming integration of Web 2.0 and SOA is supposed to focus more on "human-centered" aspects, taking humans "into-the-service-loop" and making more *collaborative* the next generation of service computing paradigms. This convergence between Web 2.0 and SOA is often cited as the *Internet of Services* (IoS), and will surely amplify the importance of more "lightweight" service computing paradigms, like the RESTful one. Therefore, future research needs to be done on how to integrate REST into current SOA solutions, mostly by automatically mediating between typical Web service technologies (WSDL, SOAP) and their REST equivalents (WADL, POX, JSON, ...). Moreover, regarding to service registry and discovery issues, novel techniques should be investigated, trying to exploit the lightweight of the services and the spreading of the Web as the main service infrastructure. Such new techniques could be extracted and developed on the basis of traditional *Web Information Retrieval* approaches, among others.

In summary, this document is a survey of some selected use cases and the state-of-the-art in the two distinct fields that will be studied in the WP. The use cases and the reviewed related work helped to focus some particular problems that can be a research topic. Starting from the issues highlighted in this document we are going to present a vision paper that will establish and refine future research directions and research tasks.

Bibliography

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, no. 36, pp. 41–50, 2003.

[2] P. Horn, "Autonomic computing: Ibm's perspective on the state of the informaiton technology." [Online]. Available: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[3] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing," *Advanced Engineering Informatics*, no. 19, pp. 181–187, 2005.

[4] "Po-jra-1.1.1: State of the art report on software engineering design knowledge and survey of hci and contextual knowledge."

[5] "Po-jra-1.2.1: State of the art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of sbas."

[6] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Physiology of the grid: An open grid services architecture for distributed systems integration," 2002. [Online]. Available: http://www.globus.org/alliance/publications/papers/ogsa.pdf

[8] I. Foster, "What is the grid? a three point checklist," 2002. [Online]. Available: http://www.gridtoday.com/02/0722/100136.html

[9] Z. Németh and V. Sunderam, "Characterizing grids: Attributes, definitions, and formalisms," *Journal of Grid Computing*, vol. 1, no. 1, 2003.

[10] J. Buisson, F. Andre, and J.-L. Pazat, "A framework for dynamic adaptation of parallel components," in *ParCo 2005*, Málaga, Spain, 13–16 September 2005. [Online]. Available: http://www.irisa.fr/paris/Biblio/Papers/Buisson/BuiAndPaz05PARCO.pdf

[11] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo, "Assist as a research framework for high-performance grid programming environments," in *Grid Computing: Software environments and Tools*, J. C. Cunha and O. F. Rana, Eds. Springer, 2005.

[12] A. Denis, C. Pérez, T. Priol, and A. Ribes, "Padico: A component-based software infrastructure for grid computing," in *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*. IEEE Computer Society, 2003.

[13] F. André, H. Bouziane, J. Buisson, J.-L. Pazat, and C. Pérez, "Towards dynamic adaptability support for the master-worker paradigm in component based applications," in *CoreGRID Symposium in conjunction with Euro-Par 2007 conference*, Rennes, France, 27-28 August 2007.

[14] P. Combes, F. Lombard, M. Quinson, and F. Suter, "A Scalable Approach to Network Enabled Servers," in *Proceedings of the 7th Asian Computing Science Conference*, January 2002, pp. 110–124.

[15] I. Foster, "I. foster," in *IFIP International Conference on Network and Parallel Computing*, ser. LNCS, no. 3779. Springer-Verlag, 2006, pp. 2–13.

[16] E. Huedo, R. S. Montero, and I. M. Llorente, "A modular meta-scheduling architecture for interfacing with pre-ws and ws grid resource management services," *Future Gener. Comput. Syst.*, vol. 23, no. 2, pp. 252–261, 2007.

[17] "Nextgrid project website." [Online]. Available: http://www.ogf.org

[18] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[19] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P. Vicat-Blanc Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet, "Grid5000: a nation wide experimental grid testbed," *in International Journal on High Performance Computing Applications*, 2006.

[20] "Po-ia-1.2.1: Definition of the collaboration infrastructure and identification of appropriate tools for exchange of knowledge."

[21] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using apples," in *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 2003.

[22] S. Vadhiyar and J. Dongarra, "GrADSolve: RPC for high performance computing on the grid," in *Euro-Par*, ser. Lecture Notes in Computer Science, vol. 2790. Springer-Verlag, 2003, parallel Processing.

[23] J. D. et Al, "Self-adapting numerical software (sans) effort," *IBM Journal of research and development*, vol. 50, no. 2/3, 2006.

[24] D. Caromel, C. Delbe, A. di Costanzo, and M. Leyton, "Proactive: an integrated platform for programming and running applications on grids and p2p systems," *Computational Methods in Science and Technology*, vol. 12, 2006. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00125034

[25] D. Caromel, "Toward a method of object-oriented concurrent programming," *Communications of the ACM*, vol. 36, no. 9, pp. 90–102, 1993.

[26] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, and C. Zoccolo, "Components for high performance grid programming in the grid.it project," in *Workshop on Component Models and Systems for Grid Applications*, June 2004.

[27] F. Galile, J.-L. Roch, G. Cavalheiro, and M. Doreille, " Athapascan-1: On-line Building Data Flow Graph in a Parallel Language," in *International Conference on Parallel Architectures and Compilation Techniques, PACT'98*, IEEE, Ed., Paris, France, Oct. 1998, pp. 88–95.

[28] S. Guelton, T. Gautier, and J.-L. Pazat, "Dynamic adaptation applied to safety," INRIA, Tech. Rep., 2008, to appear.

[29] V. Adve, V. V. Lam, and B. Ensink, "Language and compiler support for adaptive distributed applications," in *Workshop on Optimization of Middleware and Distributed Systems*. Snowbird, Utah: ACM SIGPLAN, June 2001.

[30] B. Ensink and V. Adve, "Coordinating adaptations in distributed systems," in *24th International Conference on Distributed Computing Systems*, March 2004, pp. 446–455.

[31] P.-C. David and T. Ledoux, "An aspect-oriented approach for developing self-adaptive fractal components," in *International Workshop on Software Composition (SC), Vienna, Austria*, ser. Lecture Notes in Computer Science, W. Lwe and M. Sdholt, Eds., vol. 4089. Springer Verlag, March 2006, pp. 82–97.

[32] ——, "Wildcat: a generic framework for context-aware applications," in *Proceeding of MPAC'05, the 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing*. ACM Press, November 2005, grenoble, France.

[33] L. Capra, W. Emmerich, and C. Mascolo, "CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications," *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 929–945, Oct. 2003.

[34] I. Foster, N. Jennings, and C. Kesselman, "Brain meets brawn:why grid and agents need each other," *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pp. 8–15, 2004.

[35] "Open grid forum website." [Online]. Available: http://www.ogf.org

[36] M. R. et al., "Interoperation of world-wide production e-science infrastructures," *Concurrency and Computation: Practice and Experience, OGF Special Issue*, 2008.

[37] A. Kertesz and P. Kacsuk, "Meta-broker for future generation grids: A new approach for a high-level interoperable resource management," in *CoreGRID Workshop on Grid Middleware in conjunction with ISC'07 conference, Dresden, Germany, June 25-26*, 2007.

[38] J. Seidel, O. Wldrich, P. Wieder, R. Yahyapour, and W. Ziegler, "Using sla for resource management and scheduling - a survey," in *D. Talia, R. Yahyapour and W. Ziegler editors, Grid Middleware and Services - Challenges and Solutions*. Springer, 2008.

[39] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, "Grid load balancing using intelligent agents," *Future Gener. Comput. Syst.*, vol. 21, no. 1, pp. 135–149, 2005.

[40] A. Galstyan, K. Czajkowski, and K. Lerman, "Resource allocation in the grid with learning agents," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 91–100, 2005.

[41] M. O. Shafiq, H. F. Ahmad, H. Suguri, and A. Ali, "Autonomous semantic grid: Principles of autonomous decentralized systems for grid computing," *IEICE - Trans. Inf. Syst. E88-D*, vol. 88, no. 12, pp. 2640–2650, 2005.

[42] H. U. S. Shigeta and N. Imamura, "Grid middleware for realizing autonomous resource sharing: Grid service platform," *FUJITSU SCIENTIFIC & TECHNICAL JOURNAL*, vol. 43, no. 4, 2007.

[43] W. K. Cheung, J. Liu, K. H. Tsang, and R. K. Wong, "Towards autonomous service composition in a grid environment," *icws*, vol. 0, p. 550, 2004.

[44] S. S. Vadhiyar and J. J. Dongarra, "Self adaptivity in grid computing: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, no. 2-4, pp. 235–257, 2005.

[45] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The grads project: Software support for high-level grid application development," *International Journal of High Performance Applications and Supercomputing*, vol. 15, no. 4, pp. 327–344, 2001.

[46] "Coregrid project website." [Online]. Available: http://www.coregrid.net

[47] M. Parkin, R. M. Badia, and J. Martrat, "A comparison of sla use in six of the european commissions fp6 projects," Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, Tech. Rep. TR-0129, April 2008. [Online]. Available: http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0129.pdf

[48] D. Bell, T. Kojo, P. Goldsack, S. Loughran, D. Milojicic, S. Schaefer, J. Tatemura, and P. Toft, "Configuration description, deployment, and lifecycle management (cddlm) foundation document," August 2005. [Online]. Available: http://www.ogf.org/documents/GFD.50.pdf

[49] A. Savva, H. Kishimoto, S. Newhouse, and D. Pulsipher, "Ogsa ems architecture scenarios," April 2007.

[50] S. A. et al., "Glue schema specification version 1.2," 2005.

[51] G. C. Galang, P. D. Coddington, R. Fraser, R. N. Jones, and A. Sharpe, "Experiences in deploying an mds4 grid information system on the australian national grid," in *Proceedings of APAC 2007*, 2007.

[52] F. M. Cuenca-Acuna and T. D. Nguyen, "Self-managing federated services," in *23nd IEEE International Symposium on Reliable Distributed Systems*, 2004.

[53] M. Sabou and J. Pan, "Towards semantically enhanced web service repositories," 2007.

[54] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual workspaces in the grid," ANL/MCS-P1231-0205, 2005.

[55] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "Vmplants: Providing and managing virtual machine execution environments for grid computing," in *Proceedings of the ACM/IEEE SC2004 Conference on High Performance Networking and Computing*. IEEE Computer Society, November 2004.

[56] M. Smith, T. Friese, and B. Freisleben, "Hot service deployment in an ad hoc grid environment," *ICSOC04*, 2004.

[57] A. Harrison and D. I. J. Taylor, "Dynamic web service deployment using wspeer," in *Proceedings of the 1st international workshop on Advanced data processing in ubiquitous compu*, 2006.

[58] P. Toft and S. Loughran, "Configuration description, deployment and lifecycle management working group (cddlm-wg) final report," 2008.

[59] S.-M. Yoo, J. W.-K. Hong, J.-G. Park, C.-W. Ahn, and S.-W. Kim, "Performance evaluation of wbem implementations," February 2006.

[60] H. Kreger, K. Wilson, and I. Sedukhin, "Web services distributed management: Management of web services (wsdm-mows) 1.1," web, August 2006. [Online]. Available: http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.pdf

[61] W. Vambenepe, "Services distributed management: Management using web services (muws 1.0)," web, August 2005. [Online]. Available: http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.pdf

[62] T. Banks, "Web services resource framework (wsrf) – primer v1.2," 2006.

[63] L. Qi, H. Jin, I. Foster, and J. Gawor, "Hand: Highly available dynamic deployment infrastructure for globus toolkit 4," 2006. [Online]. Available: http://www.globus.org/alliance/publications/papers/HAND-Submitted.pdf

[64] E.-K. Byun and J.-S. Kim, "Dynagrid: A dynamic service deployment and resource migration framework for wsrf-compliant applications," February 2007.

[65] A. S. Rana, F. Würthwein, K. Keahey, T. Freeman, A. Vaniachine, D. Malon, E. May, S. Martin, R. Gardner, B. Sotomayor, B. Holzman, R. Pordes, D. Skow, T. Wenaus, R. Popescu, J. Shank, E. Laure, I. Bird, M. Schulz, M. Litmaath, S. Campana, D. Smith, B. Blumenfield, K. De, M. Vranicar, P. Smith, and S. Wang, "An edge services framework (esf) for egee, lcg, and osg," 2006.

[66] E. St Pol and F. André, "A middleware for transactional hospital applications on local wireless networks," in *International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'2000)*, Las Vegas, USA, June 2000.

[67] M.-T. Segarra and F. André, "A generic approach to satisfy adaptability needs in mobile environments," in *33th Annual Hawaii International Conference on System Science (HICSS-33)*, Maui, USA, January 2000.

[68] X. Wang and H. Schulzerinne, "Comparison of adaptive internet multimedia applications," *EICE Transactions on Communications*, June 1999.

[69] D. Sisalem and H. Schulzrinne., "The loss-delay based adjustment algorithm: A tcp-friendly adaption scheme," in *Proceeding of NOSSDAV'98*, July 1998.

[70] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicas," in *Proceeding of SigComm'96*, August 1996.

[71] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer, "Adapting to network and client variation using active proxies: Lessons and perspectives." *IEEE Personal Communications, 5(4):10-19*, August 1998.

[72] O. Layaida, S. B. Atallah, and D. Hagimont, "A framework for dynamically configurable and reconfigurable network-based multimedia adaptations," *Journal of Internet Technology*, October 2004, special Issue on "Real time media delivery over the Internet.

[73] J. Keeney and V. Cahill, "Chisel: A policy-driven, context-aware, dynamic adaptation framework," in *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY' 03)*, 2003.

[74] J.-G. Kim, Y. Wang, and S.-F. Chang, "Content-adaptive utility-based video adaptation," *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, vol. 3, pp. III–281–4 vol.3, July 2003.

[75] J. D. McCarthy, M. A. Sasse, and D. Miras, "Sharp or smooth?: comparing the effects of quantization vs. frame rate for streamed video," in *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2004, pp. 535–542.

[76] L. Yan and K. Sere, "A formalism for context-aware mobile computing," in *Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, 2004.

[77] ——, "Dependability aspects of ubiquitous computing," in *Second International Conference on Availability, Reliability and Security (ARES'07)*, 2007.

[78] P. Radoslavov, R. Govindan, and D. Estrin., "Topology-informed internet replica placement," in *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, 2001.

[79] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A dynamic object replication and migration protocol for an internet hosting service," in *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 1999, pp. 101–113.

[80] S. Wu and Y. Chang, "An active replication scheme for mobile data management," in *Proceedings of the Sixth International Conference on Database Systems for Advanced Applications (DASFAA'99)*, 1999, pp. 143–150.

[81] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. V. Steen, "Replication for web hosting systems," *ACM Computing Surveys*, vol. 36, no. 3, pp. 291–334, 2004.

[82] M. T.Segarra and F.Andr, "Mfs: a mobile file system using generic system services," in *Proceedings of the 1999 ACM symposium on Applied computing*, 1999, pp. 419–420.

[83] B. Mitchell and P. Mckee, "Slas a key commercial tool," *Innovation and the Knowledge Economy: Issues, Applications, Case Studies*, 2006.

[84] D. Laforenza, F. M. Nardini, and F. Silvestri, "Collaborative ranking of grid-enabled workflow services providers," in *Proceedings of The ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC). Boston (USA). 23-27 June 2008.*, 2008.

[85] B. D., H. H., M. F., N. E., C. M., F. C., and O. D. (eds.), "Web services architecture - w3c wg note." [Online]. Available: http://www.w3.org/TR/ws-arch/

[86] J. D. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. K. Tsakalidis, "Contemporary web service discovery mechanisms," *J. Web Eng.*, vol. 5, no. 3, pp. 265–290, 2006.

[87] M. Klein and A. Bernstein, "Toward high-precision service retrieval," *Internet Computing, IEEE*, vol. 8, no. 1, pp. 30–36, Jan-Feb 2004.

[88] "Oasis uddi specifications tc - committee specifications." [Online]. Available: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[89] M. Jeckle and B. Zengler, "Active uddi - an extension to uddi for dynamic and fault-tolerant service invocation," 2002. [Online]. Available: citeseer.ist.psu.edu/jeckle02active.html

[90] A. Avila-Rosas, L. Moreau, V. Dialani, S. Miles, and X. Liu, "Agents for the grid: A comparison with web services (part ii: Service discovery)." [Online]. Available: citeseer.ist.psu.edu/538865.html

[91] OASIS/ebXML Registry Technical Committee, "OASIS/ebXML Registry Services Specification v2.5," http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf, June 2003.

[92] P. Rompothong and T. Senivongse, "A query federation of uddi registries," in *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*. Trinity College Dublin, 2003, pp. 561–566.

[93] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "A scalable and ontology-based p2p infrastructure for semantic web services." [Online]. Available: citeseer.ist.psu.edu/schlosser02scalable.html

[94] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding semantics to web services standards," 2003. [Online]. Available: citeseer.ist.psu.edu/sivashanmugam03adding.html

[95] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 161–172.

[96] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. London, UK: Springer-Verlag, 2001, pp. 329–350.

[97] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 149–160.

[98] C. Schmidt and M. Parashar, "A peer-to-peer approach to web service discovery," 2003. [Online]. Available: citeseer.ist.psu.edu/schmidt03peertopeer.html

[99] Y. Li, F. Zou, Z. Wu, and F. Ma, "Pwsd: A scalable web service discovery architecture based on peer-to-peer overlay network," in *APWeb*, ser. Lecture Notes in Computer Science, J. X. Yu, X. Lin, H. Lu, and Y. Zhang, Eds., vol. 3007. Springer, 2004, pp. 291–300.

[100] S. K., V. K., M. R., Z. Z., and S. A., "Speed-r: Semantic p2p environment for diverse web service registries." [Online]. Available: http://webster.cs.uga.edu/~mulye/SemEnt/Speed-R.html

[101] M. Montebello and C. Abela, "Daml enabled web services and agents in the semantic web," in *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*. London, UK: Springer-Verlag, 2003, pp. 46–58.

[102] "rwhois rfc." [Online]. Available: http://rfc.net/rfc2167.html

[103] L. Baresi and M. Miraz, "A distributed approach for the federation of heterogeneous registries," in *Proc. of the 4th Int'l Conf. of Service-oriented Computing*, 2006.

[104] F. Najmi (ed.), "Java API for XML Registries (JAXR)," http://java.sun.com/webservices/jaxr/, October 2002.

[105] A. Tsalgatidou, M. Pantazoglou, and G. Athanasopoulos, "Specification of the Unified Service Query Language (USQL)," June 2006, Tech. Rep.

[106] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[107] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval, Tech. Rep. UT-CS-94-270, 1994. [Online]. Available: citeseer.ist.psu.edu/berry95using.html

[108] C. Platzer and S. Dustdar, "A vector space search engine for web services," in *ECOWS*. IEEE Computer Society, 2005, pp. 62–71.

[109] K. Zachos, N. A. M. Maiden, X. Zhu, and S. Jones, "Discovering web services to specify more complete system requirements," in *CAiSE*, ser. Lecture Notes in Computer Science, J. Krogstie, A. L. Opdahl, and G. Sindre, Eds., vol. 4495. Springer, 2007, pp. 142–157.

[110] M. N. and Z. K., "Inventing requirements from software: An empirical investigation with web services."

[111] X. Dong, J. Madhavan, and A. Halevy, "Mining structures for semantics," *SIGKDD Explor. Newsl.*, vol. 6, no. 2, pp. 53–60, 2004.

[112] A. Zaremski and J. Wing, "Signature matching: a tool for using software libraries," *ACM Trans. Softw. Eng. Methodol.*, vol. 4, no. 2, pp. 146–170, 1995.

[113] ——, "Specification matching of software components," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 4, pp. 333–369, 1997.

[114] E. Stroulia and Y. Wang, "Structural and semantic matching for assessing Web-service similarity," *Int'l J. Cooperative Inf. Syst.*, vol. 14, no. 4, pp. 407–438, 2005.

[115] D. Bianchini, V. D. Antonellis, B. Pernici, and P. Plebani, "Ontology-based methodology for e-service discovery," *Journal on Information Systems*, vol. 31, no. 4-5, pp. 361–380, 2006.

[116] M. Crasso, A. Zunino, and M. Campo, "Query by example for web services," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2008, pp. 2376–2380. [Online]. Available: http://portal.acm.org/citation.cfm?id=1363686.1364251

[117] A. Zisman, K. Mahbub, and G. Spanoudakis, "A service discovery framework based on linear composition," *Services Computing, 2007. SCC 2007. IEEE International Conference on*, pp. 536–543, July 2007.

[118] I. Budak Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko, "Ontology-driven Web services composition platform," in *Proc. IEEE Int'l Conference on e-Commerce Technology (CEC'04)*, 6-9 July 2004, pp. 146–152.

[119] M. Mecella, B. Pernici, and P. Craca, "Compatibility of e-services in a cooperative multi-platform environment," in *Proc. Int'l Workshop on Technologies for E-Services (TES'01), Rome, Italy, September 14-15*, 2001, pp. 44–57.

[120] R. Akkiraju and B. Sapkota, "Semantic Annotations for WSDL and XML Schema. Usage Guide," http://www.w3.org/TR/sawsdl-guide/, January 2007.

[121] "Owl-s specifications." [Online]. Available: http://www.daml.org/services/owl-s/1.0/

[122] H. Lausen, A. Polleres, and D. Roman (eds.), "Web Services Modeling Ontology (WSMO). W3C Submission," http://www.w3.org/Submission/WSMO/, June 2005.

[123] K. Sycara, S. Widoff, M. Klusch, and J. Lu, "Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace," *Autonomous Agents and Multi-Agent Systems*, vol. 5, no. 2, pp. 173–203, 2002.

[124] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic matching of web services capabilities," in *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*. Sardinia, Italy: Springer-Verlag, 2002, pp. 333–347.

[125] L. Li and I. Horrocks, "A software framework for matchmaking based on semantic web technology," *International Journal of Electronic Commerce*, vol. 8, no. 4, pp. 39–60, 2004.

[126] V. Haarslev and R. Möller, "Racer: A core inference engine for the semantic web," in *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20*, 2003, pp. 27–36.

[127] L. Moreau, S. Miles, J. Papay, K. Decker, and T. Payne, "An integrative approach for attaching semantic annotations to service descriptions," in *Proceedings of the International Workshop on Grid Information Services in ISWC 2003*. Edinburgh, Scotland: IEEE Computer Society, 2003.

[128] R. Akkiraju, R. Goodwin, P. Doshi, and S. Roeder, "A method for semantically enhancing the service discovery capabilities of uddi," in *IIWeb*, S. Kambhampati and C. A. Knoblock, Eds., 2003, pp. 87–92.

[129] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Importing the semantic web in uddi," in *CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*. Toronto, Canada: Springer-Verlag, 2002, pp. 225–236.

[130] T. Syeda-Mahmood, G. Shah, R. Akkiraju, A. Ivan, and R. Goodwin, "Searching service repositories by combining semantic and ontological matching," in *Int'l Conf. on Web Services (ICWS'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 13–20.

[131] A. Dogac, Y. Kabak, and G. B. Laleci, "Enriching ebxml registries with owl ontologies for efficient service discovery," in *RIDE '04: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*. Boston, USA: IEEE Computer Society, 2004, pp. 69–76.

[132] H.-K. Yu, T.-M. Tsai, Y.-F. Yang, and W.-S. Huang, "Ontology-mediated service matching and adaptation," in *OTM Workshops*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 2889. Springer, 2003, pp. 197–209.

[133] I. Elgedawy, "A conceptual framework for web services semantic discovery," in *OTM Workshops*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 2889. Springer, 2003, pp. 1004–1016.

[134] T. Kawamura, J.-A. D. Blasio, T. Hasegawa, M. Paolucci, and K. P. Sycara, "Preliminary report of public experiment of semantic service matchmaker with uddi business registry," in *ICSOC*, ser. Lecture Notes in Computer Science, M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, Eds., vol. 2910. Springer, 2003, pp. 208–224.

[135] T. D. Noia, E. D. Sciascio, F. M. Donini, and M. Mongiello, "A system for principled matchmaking in an electronic marketplace," *Int. J. Electron. Commerce*, vol. 8, no. 4, pp. 9–37, 2004.

[136] A. Borgida and P. F. Patel-Schneider, "A semantics and complete algorithm for subsumption in the classic description logic," *Journal of Artificial Intelligence Research (JAIR)*, vol. 1, pp. 277–308, 1994.

[137] B. Benatallah, M.-S. Hacid, C. Rey, and F. Toumani, "Request rewriting-based web service discovery," in *International Semantic Web Conference*, ser. Lecture Notes in Computer Science, D. Fensel, K. P. Sycara, and J. Mylopoulos, Eds., vol. 2870. Springer, 2003, pp. 242–257.

[138] W.-T. Balke and M. Wagner, "Cooperative discovery for user-centered web service provisioning," in *ICWS*, L.-J. Zhang, Ed. CSREA Press, 2003, pp. 191–197.

[139] ——, "Towards personalized selection of web services," in *WWW (Alternate Paper Tracks)*, 2003.

[140] Z. Hu, "Using ontology to bind web services to the data model of automation systems revised papers from the node 2002 web and database-related workshops on web," 2002. [Online]. Available: citeseer.ist.psu.edu/hu02using.html

[141] S. Overhage and P. Thomas, "Ws-specification: Specifying web services using uddi improvements." [Online]. Available: citeseer.ist.psu.edu/overhage02wsspecification.html

[142] J. Ma, Y. Zhang, and J. He, "Efficiently finding web services using a clustering semantic approach," in *CSSSIA*, ser. ACM International Conference Proceeding Series, Q. Z. Sheng, U. Nambiar, A. P. Sheth, B. Srivastava, Z. Maamar, and S. Elnaffar, Eds., vol. 292. ACM, 2008, p. 5.

[143] S. Dietze, A. Gugliotta, and J. Domingue, "Towards context-aware semantic web service discovery through conceptual situation spaces," in *CSSSIA*, ser. ACM International Conference Proceeding Series, Q. Z. Sheng, U. Nambiar, A. P. Sheth, B. Srivastava, Z. Maamar, and S. Elnaffar, Eds., vol. 292. ACM, 2008, p. 6.

[144] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dept. of Computer Science, Dartmouth College, Tech. Rep. TR2000-381, November 2000. [Online]. Available: citeseer.ist.psu.edu/chen00survey.html

[145] "Codamos." [Online]. Available: http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/CoDAMoS/ontology/

[146] P. D. and et al., "Towards an extensible context ontology for ambient intelligence."

[147] S. T. and L.-P. C., "A context modelling survey." [Online]. Available: http://citeseer.ist.psu.edu/712164.html

[148] T. Strang, C. Linnhoff-Popien, and K. Frank, "CoOL: A Context Ontology Language to enable Contextual Interoperability," in *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, ser. Lecture Notes in Computer Science (LNCS), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893. Paris/France: Springer Verlag, November 2003, pp. 236–247. [Online]. Available: citeseer.ist.psu.edu/strang03cool.html

[149] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui, "Toward an agent-based and context-oriented approach for web services composition," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 5, pp. 686–697, 2005.

[150] G. Spanoudakis, K. Mahbub, and A. Zisman, "A platform for context aware runtime web service discovery," in *ICWS*. IEEE Computer Society, 2007, pp. 233–240.

[151] Z. A., D. J., and S. G., "Proactive runtime service discovery."

[152] S. Ran, "A model for web service discovery with qos," *ACM SIGecom Exchanges*, vol. 4, no. 1, pp. 1–10, Spring 2003.

[153] G. von Bochmann, B. Kerhervé, H. Lutfiyya, M.-V. M. Salem, and H. Ye, "Introducing qos to electronic commerce applications," in *ISEC '01: Proceedings of the Second International Symposium on Topics in Electronic Commerce*. London, UK: Springer-Verlag, 2001, pp. 138–147.

[154] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality driven web services composition," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM, 2003, pp. 411–421.

[155] M. Conti, E. Gregori, and F. Panzieri, "Load distribution among replicated web servers: a qos-based approach," *SIGMETRICS Perform. Eval. Rev.*, vol. 27, no. 4, pp. 12–19, 2000.

[156] O. Ardaiz, F. Freitag, and L. Navarro, "Improving the service time of web clients using server redirection," *SIGMETRICS Perform. Eval. Rev.*, vol. 29, no. 2, pp. 39–44, 2001.

[157] M. Merzbacher and D. Patterson, "Measuring end-user availability on the web: practical experience," *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 473–477, 2002.

[158] A. van Moorsel, "Metrics for the internet age: Quality of experience and quality of business." [Online]. Available: citeseer.ist.psu.edu/vanmoorsel01metrics.html

[159] V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian, "A quality of service management framework based on user expectations." in *ICSOC*, ser. Lecture Notes in Computer Science, M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, Eds., vol. 2910. Springer, 2003, pp. 104–114.

[160] S. Frølund and J. Koistinen, "Quality of services specification in distributed object systems design," *COOTS'98: Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, vol. 5, no. 4, pp. 179–202, 1998.

[161] M. Fugini, P. Plebani, and F. Ramoni, "A user driven policy selection model," in *ICSOC '06: Proc. of the 4th Int'l Conf. on Service oriented computing.*, 2006.

[162] C. Cappiello, M. Comuzzi, and P. Plebani, "On Automated Generation of Web Service Level Agreements," in *The 19th International Conference on Advanced Information Systems Engineering (CAiSE'07).*, Trondheim, Norway, June 2007.

[163] C. Zhou, L.-T. Chia, and B.-S. Lee, "Daml-qos ontology for web services," in *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*. San Diego, CA, USA: IEEE Computer Society, 2004, pp. 472–479.

[164] A. R. Cortés, O. Martín-Díaz, A. D. Toro, and M. Toro, "Improving the automatic procurement of web services using constraint programming," *International Journal of Cooperative Information Systems*, vol. 14, no. 4, pp. 439–468, 2005.

[165] P. Van Hentenryck and V. Saraswat, "Strategic directions in constraint programming," *ACM Computing Surveys*, vol. 28, no. 4, pp. 701–726, 1996.

[166] K. Kritikos and D. Plexousakis, "Semantic qos metric matching," in *ECOWS '06: Proceedings of the European Conference on Web Services*. Zurich, Switzerland: IEEE Computer Society, 2006, pp. 265–274.

[167] ——, "Semantic qos-based web service discovery algorithms," in *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*. Halle, Germany: IEEE Computer Society, 2007, pp. 181–190.

[168] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel, "Specifying and monitoring guarantees in commercial grids through sla," in *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2003, p. 292.

[169] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati, "Automated sla monitoring for web services," in *DSOM '02: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. Springer-Verlag, 2002, pp. 28–41. [Online]. Available: http://portal.acm.org/citation.cfm?id=645354.650585

[170] P. Bhoj, S. Singhal, and S. Chutani, "Sla management in federated environments," *Comput. Networks*, vol. 35, no. 1, pp. 5–24, 2001.

[171] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1999, pp. 24–35.

[172] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail, "G-qosm: Grid service discovery using qos properties," *Computers and Artificial Intelligence*, vol. 21, no. 4, 2002.

[173] S. Miles, J. Papay, V. Dialani, M. Luck, K. Decker, T. Payne, and L. Moreau, "Personalised grid service discovery," *Software, IEE Proceedings* -, vol. 150, no. 4, pp. 252–256, 26 Aug. 2003.

[174] D. Xu, K. Nahrstedt, and D. Wichadakul, "Qos-aware discovery of wide-area distributed services," in *Proc. of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '01)*, 2001.

[175] A. Michlmayr, F. Rosenberg, C. Platzer, and S. Dustar, "Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective," in *Proceedings of the International Workshop on Service Oriented Software Engineering (IW-SOSE'07)*, 2007.

[176] M. D. Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. D. Nitto, "WS Binder: a framework to enable dynamic binding of composite web services," in *Proceedings of the International Workshop on Service-Oriented Software Engineering (SOSE '06)*. New York, NY, USA: ACM, 2006, pp. 74–80.

[177] C. Pautasso and G. Alonso, "Flexible Binding for Reusable Composition of Web Services," in *Software Composition*, 2005, pp. 151–166.

[178] M. J. Duftler, N. K. Mukhi, A. Slominski, and S. Weerawarana, "Web Services Invocation Framework (WSIF)," in *Proceedings of the OOPSLA Workshop on Object-Oriented Web Services*, 2001.

[179] P. Leitner, F. Rosenberg, and S. Dustdar, "Daios – Efficient Dynamic Web Service Invocation," Vienna University of Technology, Tech. Rep. TUV-1841-2007-01, 2007.

[180] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani, "Developing Adapters for Web Services Integration," in *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE 2005)*, ser. Lecture Notes in Computer Science, 2005.

[181] E. Cimpian, A. Mocan, and M. Stollberg, "Mediation Enabled Semantic Web Services Usage," in *Proceedings of the Asian Semantic Web Conference (ASWC)*, vol. 4185, 2006, pp. 459–473.

[182] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "End-to-End Versioning Support for Web Services," in *Proceedings of the International Conference on Services Computing (SCC 2008)*. IEEE Computer Society, July 2008.

[183] K. Brown and M. Ellis, "Best practices for Web services versioning," 2004, http://www-128.ibm.com/developerworks/webservices/library/ws-version/ (Last Accessed: June 13th, 2008).

[184] P. Kaminski, H. Müller, and M. Litoiu, "A Design for Adaptive Web Service Evolution," in *Proceedings of the International Workshop on Self-Adaptation and Self-Managing Systems (SEAMS '06)*. New York, NY, USA: ACM, 2006, pp. 86–92.

[185] M. Treiber, H.-L. Truong, and S. Dustdar, "SEMF - - Service Evolution Management Framework," in *Proceedings of the EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Special session on Quality and Service-Oriented Applications*, 2008.

[186] J. Kephart, "Research challenges of autonomic computing," *International Conference on Software Engineering: Proceedings of the 27 th international conference on Software engineering*, vol. 15, no. 21, pp. 15–22, 2005.

[187] Q. Z. Sheng, U. Nambiar, A. P. Sheth, B. Srivastava, Z. Maamar, and S. Elnaffar, Eds., *Proceedings of the 2008 International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation: organized with the 17th International World Wide Web Conference (WWW 2008), CSSSIA 2008, Beijing, China, April 22, 2008*, ser. ACM International Conference Proceeding Series, vol. 292. ACM, 2008.

[188] M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, Eds., *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2910. Springer, 2003.

[189] R. Meersman and Z. Tari, Eds., *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, OTM Confederated International Workshops, HCI-SWWA, IPW, JTRES, WORM, WMS, and WRSM 2003, Catania, Sicily, Italy, November 3-7, 2003, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2889. Springer, 2003.